24th EACSL Annual Conference on Computer Science Logic

CSL 2015, September 7–10, 2015, Berlin, Germany

Edited by

Stephan Kreutzer

LIPIcs – Vol. 41 – CSL 2015 www.dagstuhl.de/lipics

Editor

Stephan Kreutzer
Technische Universität Berlin
Ernst-Reuter Platz 7
10587 Berlin, Germany
stephan.kreutzer@tu-berlin.de

ACM Classification 1998
A.0 Conference Proceedings, C.2.4 Distributed Systems, D.2.4 Software/ Programs Verifications, D.3.1 Formal Definitions and Theory, D.3.3 Languages Constructs and Features, I.2.4 Knowledge Representations Formalisms and Methods, F Theory of Computation, F.4.1 Mathematical Logic

ISBN 978-3-939897-90-3

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at http://www.dagstuhl.de/dagpub/978-3-939897-90-3.

Publication date
September, 2015

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at http://dnb.d-nb.de.

Digital Object Identifier: 10.4230/LIPIcs.CSL.2015.i

ISBN 978-3-939897-90-3 **ISSN 1868-8969** **http://www.dagstuhl.de/lipics**

LIPIcs – Leibniz International Proceedings in Informatics

LIPIcs is a series of high-quality conference proceedings across all fields in informatics. LIPIcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

ISSN 1868-8969

http://www.dagstuhl.de/lipics

Contents

24th EACSL Annual Conference on Computer Science Logic (CSL 2015).
Editor: Stephan Kreutzer
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
EACSL EUROPEAN ASSOCIATION FOR COMPUTER SCIENCE LOGIC

Preface

The annual conference *Computer Science Logic* (CSL 2015) of the European Association for Computer Science Logic (EACSL) was held in Berlin, Germany, from 7 to 10 September 2015. CSL started as a series of international workshops on Computer Science Logic, and became at its sixth meeting the Annual Conference of the EACSL. This conference was the 29th workshop and 24th EACSL conference. The conference was organised by the Logic and Semantics Research Group of the Technical University Berlin.

A total of 99 abstracts were registered for the conference. After a two week electronic meeting, the programme committee selected 39 papers for presentation at the conference and publication in the proceedings. Each paper was assigned to at least three programme committee members. The overall quality of the submissions was very high with essentially no bad papers submitted. This made the work of the programme committee a difficult task. Due to lack of space, the programme committee had to reject several very good papers in the end.

In 2015, CSL followed a selective rebuttal strategy. There was no general rebuttal phase but whenever questions arose during the discussion or in a review that could meaningfully be posed to the authors, the authors were notified and asked for clarification. The average response time by the authors was less than a day, even for difficult questions, making this a valuable tool for the programme committee discussion. As a result of this selective rebuttal system, two papers were withdrawn by the authors due to flaws in their arguments that could not be fixed. For several other papers, the referees' concerns could be clarified by the authors or the authors were able to present simple fixes for factual mistakes in their papers which were verified and accepted by the programme committee.

The programme committee was assisted by a number of external reviewers providing additional expertise. The list of external reviewers is included in these proceedings. On behalf of the programme committee I wish to express my sincere gratitude to the external referees for the time and energy they spent on assessing submissions to CSL 2015.

In addition to the contributed talks, CSL 2015 had four invited speakers:

- Martín Abadi (Google),
- Elham Kashefi (Edinburgh),
- Markus Lohrey (Siegen), and
- Ryan Williams (Stanford).

Some invited speakers have contributed an abstract which is included in the proceedings.

The Ackermann Award is the EACSL Outstanding Dissertation Award for Logic in Computer Science. This year, the eleventh Ackermann Award was presented at CSL 2015. The jury decided to give the Ackermann Award for 2015 to Hugo Férée and Mickael Randour. The awards were officially presented at the conference on 9 September 2015. The citation of the awards, an abstract of the theses and a biographical sketch of the recipients written by Anuj Dawar is included in the proceedings.

I wish to warmly thank all members of the programme committee and all external reviewers for the time and energy spent on reviewing and discussing the papers.

Very special thanks go to Christoph Dittmann who collected the papers from the authors and compiled them into these proceedings, solving numerous LaTeX-issues and checking the papers for layout consistency. Many thanks also to Marc Herbstritt from the Dagstuhl/LIPIcs team for assisting us in the publication process and the final production of the proceedings.

Finally, I also want to thank the members of the organising committee, especially

24th EACSL Annual Conference on Computer Science Logic (CSL 2015).
Editor: Stephan Kreutzer
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
EACSL EUROPEAN ASSOCIATION FOR COMPUTER SCIENCE LOGIC

Christoph Dittmann, Jana Pilz, Roman Rabinovich and Sebastian Siebertz, for their proactive, thoughtful, reliable and energetic help in organising CSL 2015.

The conference received support from the Technical University Berlin, from the European Association for Computer Science Logic (EACSL) and from the Deutsche Forschungsgemeinschaft (DFG). I thank these organisations for their generous support.

Stephan Kreutzer

Conference Organisation

Programme Committee

- Albert Atserias (Universitat Politècnica de Catalunya, Barcelona)
- Achim Blumensath (Technical University Darmstadt)
- Mikolaj Bojanczyk (Warsaw University)
- Maria Paola Bonacina (Università degli Studi di Verona)
- Patricia Bouyer-Decitre (LSV, CNRS & ENS de Cachan)
- Ugo Dal Lago (Università di Bologna)
- Maribel Fernández (King's College London)
- Richard Garner (Macquarie University, Sydney)
- Rajeev Goré (Australian National University, Canberra)
- Stéphane Graham-Lengrand (CNRS & École Polytechnique, Paris-Saclay)
- Martin Grohe (RWTH Aachen University)
- Lauri Hella (University of Tampere)
- Martin Hofmann (Ludwig-Maximilians-University Munich)
- Stephan Kreutzer (Technical University Berlin, PC chair)
- Martin Lange (University of Kassel)
- Luigi Santocanale (LIF, Aix-Marseille Université & CNRS)
- Alexandra Silva (Radboud University Nijmegen)
- Alex Simpson (University of Edinburgh)
- Sonja Smets (University of Amsterdam)
- Makoto Tatsuta (National Institute of Informatics, Tokyo)
- Kazushige Terui (Kyoto University)
- James Benjamin Worrell (University of Oxford)
- Nobuko Yoshida (Imperial College London)

Organising Committee

- Saeed Amiri (Technical University Berlin)
- Christoph Dittmann (Technical University Berlin)
- Viktor Engelmann (Technical University Berlin)
- Stephan Kreutzer (Technical University Berlin, chair)
- Jana Pilz (Technical University Berlin)
- Roman Rabinovich (Technical University Berlin)
- Sebastian Siebertz (Technical University Berlin)

24th EACSL Annual Conference on Computer Science Logic (CSL 2015).
Editor: Stephan Kreutzer
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
EACSL EUROPEAN ASSOCIATION FOR COMPUTER SCIENCE LOGIC

External Reviewers

Alexis Bernadet
Amaldev Manuel
Andreas Herzig
Andrew M. Marshall
Antoine Miné
Antti Kuusisto
Arnaud Carayol
Arnaud Durand
Arnaud Sangnier
Ashutosh Trivedi
Ben Moszkowski
Benedetto Intrigila
Benedikt Bollig
Beniamino Accattoli
Bernhard Reus
Charles Grellois
Charles Paperman
Christof Löding
Daisuke Kimura
Damiano Mazza
Dan Ghica
Daniel Kernberger
Daniele Varacca
Detlef Plump
Diego Figueira
Dimitrios Vytiniotis
Dominic Orchard
Dominique Larchey-Wendling
Erich Grädel
Felix Canavoi
Florian Bruse
Gabriele Pulcini
Georg Zetzsche
Igor Walukiewicz
Ilya Shapirovsky
Iosif Petrakis
Jakob Rehof
James Brotherston
Jelena Ivetic
Jérôme Fortier
Joanna Ochremiak
Joel Ouaknine
Johannes Waldmann
Jonathan Hayman
Jordi Levy
Joshua Moerman
Juha Kontinen
Jules Hedges
Julian Bitterlich
Julian Bradfield
Julian Gutierrez
Kerkko Luosto
Koji Nakazawa
Kord Eickmeyer
Louwe B. Kuijer
Manuel Bodirsky
Marcus Kracht
Marino Miculan
Martin Avanzini
Martin Otto
Matthew Collinson
Michael Vanden Boom
Milka Hutagalung
Minghui Ma
Mnacho Echenim
Nicolas Markey
Nicole Schweikardt
Nikos Tzevelekos
Philipp Ruemmer
Ramyaa Ramyaa
Roy Dyckhoff
Sebastian Siebertz
Standa Zivny
Stefan Göller
Stefan Haar
Stefan Kiefer
Stephan Merz
Stéphane Demri
Takeshi Tsukada
Thomas Colcombet
Uwe Waldmann
Vincent Aravantinos
Vincent Rahli
Wouter M. Koolen
Zhe Hou

The Ackermann Award 2015

Anuj Dawar, Dexter Kozen, and Simona Ronchi Della Rocca

Members of the Jury of the EACSL Ackermann Award

Abstract

The eleventh Ackermann Award is presented at CSL'15 in Berlin, Germany. This year, again, the EACSL Ackermann Award is generously sponsored by the Kurt Gödel Society. Besides providing financial support for the Ackermann Award, the Kurt Gödel Society has also committed to inviting the recipients of the Award for a special lecture to be given to the Society in Vienna.

The 2015 Ackermann Award was open to PhD dissertations in topics specified by the CSL and LICS conferences, which were formally accepted as theses for the award of a PhD degree at a university or equivalent institution between 1 January 2013 and 31 December 2014. The Jury received ten nominations for the Ackermann Award 2015. The candidates came from a number of different countries across the world. The institutions at which the nominees obtained their doctorates represent eight countries in Europe, North America, and the Middle East.

The topics covered a wide range of Logic and Computer Science as represented by the LICS and CSL Conferences. All submissions were of a very high standard and contained remarkable contributions to their particular fields. The Jury wishes to extend its congratulations to all nominated candidates for their outstanding work. The Jury encourages them to continue their scientific careers and hopes to see more of their work in the future.

The task of the jury proved very difficult, and opinions were divided. In the end, the jury decided to award the **2015 Ackermann Award** jointly to two dissertations. The winners are (in alphabetical order):

- Hugo Férée from France, for his thesis
  Complexité d'ordre supérieur et analyse récursive
  approved by the Université de Lorraine, France, in 2014,
  supervised by Jean-Yves Marion and Mathieu Hoyrup; and
- Mickael Randour from Belgium, for his thesis
  Synthesis in Multi-Criteria Quantitative Games
  approved by the Université de Mons, Belgium, in 2014,
  supervised by Véronique Bruyère and Jean-François Raskin.

Digital Object Identifier 10.4230/LIPIcs.CSL.2015.xv

1 Hugo Férée

Citation. Hugo Férée receives the *2015 Ackermann Award* of the European Association of Computer Science Logic (EACSL) for his thesis

Complexité d'ordre supérieur et analyse récursive.

His thesis establishes deep and original results related to a variety of topics in the complexity of analytic and higher-order functions. By identifying the intrinsic limitations of the traditional approaches, they enable one to see beyond them. They provide a higher-order complexity theory which has been lacking and will certainly be recognized and used in the future. A brief survey of the content of the thesis follows, organized thematically.

24th EACSL Annual Conference on Computer Science Logic (CSL 2015).
Editor: Stephan Kreutzer; pp. xv–xviii

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Complexity of higher order real functionals. The study of computability over real numbers dates back to Turing's 1936 paper and has been an active area of research ever since. In the early 1980s serious work began on studying resource-bounded computability over the reals. That is, understanding what is and is not feasibly computable over the reals for various notions of computational feasibility. Férée starts from the results of Kawamura and Cook about functionals over $\mathtt{C}[0,1]$, the space of continuous functions over the unit interval. Namely he studies the case of norms over real functions defined on such a space and obtains several results allowing us to understand how imposing restrictions on the computational resources of these functionals affects their analytical properties. He introduces a new analytical notion expressing that the norm of a function highly depends on the value of the function at a point. He relates this analytical notion to computability and complexity: if a norm highly depends on a point then in order to compute the norm of a function, this function has to be evaluated at this point. In particular if a norm is efficiently computable then it cannot depend on too many points.

Complexity in analysis. The extension of computability theory from the natural numbers to objects coming from mathematical analysis is now well understood. However complexity theory is more problematic, but some extensions are available. Kawamura and Cook developed complexity theory for spaces that can be suitably represented by functions from finite strings to finite strings (functions of order type 1). The problem is then to understand when this approach can be used. Férée obtains results identifying a topological property of the space that is necessary for this approach to work. More precisely, if the space $\mathtt{C}(\mathtt{X},\mathtt{R})$ of continuous functions from a space $\mathtt{X}$ to the real numbers admits a representation making the complexity of the evaluation, from $\mathtt{C}(\mathtt{X},\mathtt{R}) \times \mathtt{X} \to \mathtt{R}$, well-defined, then $\mathtt{X}$ must be σ-compact. This result suggests that using higher order functions to represent objects may help in extending complexity notions to other spaces.

Higher order complexity. Until now, no satisfactory notion of complexity at higher-types has been given. Classical complexity theory provides notions of complexity for functions from $\mathtt{N} \to \mathtt{N}$ (order 1 functions, for instance the class `FPTIME`) as well as functions from $\mathtt{N} \to \mathtt{N}$ to $\mathtt{N} \to \mathtt{N}$ (order 2 functions, for instance the class `BFF`). The class `BFF` is defined for any finite type, however it is not a complexity class in the sense that it is not defined as a class of functions computable with limited resources (such as polynomial time). One reason for that is that there is no obvious notion of size for inputs, when they are themselves higher order functionals. Férée makes use of game semantics in order to represent the computation of higher order functionals and to measure their size and complexity. More precisely, he uses as inputs strategies in some sequential games whose size is supplied through a computation model called a higher-order Turing machine, consisting of a Turing machine that dialogues with its oracle describing the input. Then, using higher-order polynomials, a higher-order analogue of the usual polynomials, he defines a class of polynomial-time computable strategies. Finally, he applies this framework to the games used to solve the full abstraction problem for the class of higher-order computable PCF-functions, so supplying a notion of complexity over PCF.

Biographical Sketch. Hugo Férée was born on 20 September 1988 in Nancy. After completing his schooling there, he was a student at the École Normale Supérieure in Lyon from 2008, obtaining a Master's degree in Theoretical Computer Science in 2011, awarded jointly with the Université Claude Bernard. He then returned to Lorraine to pursue a PhD, which

he successfully defended in December 2014. He currently holds a post-doctoral position at the Technische Universität Darmstadt in Germany.

2 Mickael Randour

Citation. Mickael Randour receives the *2015 Ackermann Award* of the European Association of Computer Science Logic (EACSL) for his thesis

Synthesis in Multi-Criteria Quantitative Games.

His thesis contains important theoretical and practical contributions to the analysis of quantitative games and the synthesis of winning strategies. The contributions are technically challenging and practically relevant. Several notions introduced in the thesis significantly advance the state of the art and open important new research perspectives.

Background. Reactive systems are computer systems that maintain a continuous interaction with their environment. Examples of reactive systems include controllers embedded in cars and aeroplanes, computer system device drivers, and communication protocols in networks. Producing reactive systems that behave correctly is a notoriously challenging problem due to factors such as concurrency, uncertainty, and real-time constraints. Moreover, their correctness is often critical, as they frequently appear in contexts in which lives are at stake. A reactive system is typically modelled as a game between two players, the system and its environment. The correctness of the system is then expressed in terms of a winning condition in the game. Often these winning conditions involve quantitative measures such as mean-payoff, total-payoff, or energy constraints. The analysis of such games is notoriously difficult.

Contributions of the thesis. Randour makes a number of contributions that advance our understanding of the quantitative games, the algorithms for solving them, and the complexity of the associated decision problems. In doing so, he introduces innovative concepts that can have a lasting impact on the research field.

One important innovation is the first analysis of multiple simultaneous objectives. Another is the introduction and analysis of so-called window objectives, a conservative approximation of the standard mean-payoff and total-payoff objectives. It is shown that window objectives provide a conservative approximation of the classical mean-payoff and total-payoff measures, with complexities that break the previous barriers. Finally, the thesis studies tradeoffs between traditional worst-case and expected-case analysis. Previously, classical games typically involve an environment which is purely antagonistic and ask for strict guarantees, whereas stochastic models attempt to optimize the expected payoff, with no guarantee on individual outcomes. In this thesis, Randour has shown that one can find a reasonable trade-off between these aspects: good expected performance in the everyday situations while ensuring a strict performance threshold even in the event of unlikely worst-case circumstances.

Biographical Sketch. Mickael Randour was born on 9 July 1984 in La Louvière. All of his higher education was completed at the Université de Mons, where he obtained a Bachelor's degree in 2008, a Master's degree in 2010 and a PhD in April 2014, all of them in Computer Science. In 2010 he was the recipient of an award from the *Fondation Emile Cornez* for the best graduating student from the university. He is currently a postdoctoral researcher at the École Nomrale Supérieure de Cachan in France.

3 Jury

The Jury for the **Ackermann Award 2015** consisted of eight members, two of them *ex officio*, namely, the president and the vice-president of EACSL. This year, for the first time, the jury also included a representative of SigLog (the ACM Special Interest Group on Logic and Computation).

The members of the jury were:

- Thierry Coquand (Chalmers University of Gothenburg),
- Anuj Dawar (University of Cambridge), the president of EACSL,
- Dexter Kozen (Cornell University), SigLog representative,
- Orna Kupferman (Hebrew University of Jerusalem),
- Daniel Leivant (Indiana University, Bloomington),
- Luke Ong (University of Oxford),
- Jean-Éric Pin (CNRS and Université Paris 7),
- Simona Ronchi Della Rocca (University of Torino), the vice-president of EACSL.

3.1 Previous winners

Previous winners of the Ackermann Award were:

2005, Oxford:
Mikołaj Bojańczyk from Poland,
Konstantin Korovin from Russia, and
Nathan Segerlind from the USA.

2006, Szeged:
Balder ten Cate from The Netherlands, and
Stefan Milius from Germany.

2007, Lausanne:
Dietmar Berwanger from Germany and Romania,
Stéphane Lengrand from France, and
Ting Zhang from the People's Republic of China.

2008, Bertinoro:
Krishnendu Chatterjee from India.

2009, Coimbra:
Jakob Nordström from Sweden.

2010, Brno:
no award given.

2011, Bergen:
Benjamin Rossman from USA.

2012, Fontainebleau:
Andrew Polonsky from Ukraine, and
Szymon Toruńczyk from Poland.

2013, Turin:
Matteo Mio from Italy.

2014, Vienna:
Michael Elberfeld from Germany.

Detailed reports on their work appeared in the CSL proceedings and are also available on the EACSL homepage.

The Prophecy of Timely Rollback*

Martín Abadi

Google
Mountain View, California, USA

Abstract

Techniques for rollback recovery play a central role in ensuring fault-tolerance in many distributed systems [5]. This talk addresses the formal specification and analysis of those techniques. In particular, we will discuss the relevance of prophecy variables [4] (auxiliary program variables whose values are defined in terms of current program state and future behavior) to reasoning about systems with undo operations [1]. We will then focus on a model for data-parallel computation with a notion of virtual time [6, 2]. In this model, rollbacks allow the selective undo of work at particular virtual times [3]. A refinement theorem ensures the consistency of rollbacks.

This talk is largely based on joint work with Michael Isard.

1998 ACM Subject Classification F.3.1 Specifying and Verifying and Reasoning about Programs

Keywords and phrases Dataflow, refinement, rollback

Digital Object Identifier 10.4230/LIPIcs.CSL.2015.1

Category Invited Talk

References

1 Martín Abadi. The prophecy of undo. In Alexander Egyed and Ina Schaefer, editors, *Fundamental Approaches to Software Engineering – 18th International Conference, FASE 2015, Proceedings*, pages 347–361. Springer, 2015.

2 Martín Abadi and Michael Isard. Timely dataflow: A model. In Susanne Graf and Mahesh Viswanathan, editors, *Formal Techniques for Distributed Objects, Components, and Systems – 35th IFIP WG 6.1 International Conference, FORTE 2015, Proceedings*, pages 131–145. Springer, 2015.

3 Martín Abadi and Michael Isard. Timely rollback: Specification and verification. In Klaus Havelund, Gerard Holzmann, and Rajeev Joshi, editors, *NASA Formal Methods – 7th International Symposium, NFM 2015, Proceedings*, pages 19–34. Springer, 2015.

4 Martín Abadi and Leslie Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 82(2):253–284, 1991.

5 E. N. Elnozahy, Lorenzo Alvisi, Yi-Min Wang, and David B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Surveys*, 34(3):375–408, 2002.

6 Derek Gordon Murray, Frank McSherry, Rebecca Isaacs, Michael Isard, Paul Barham, and Martín Abadi. Naiad: a timely dataflow system. In *ACM SIGOPS 24th Symposium on Operating Systems Principles*, pages 439–455, 2013.

* Most of this work was done at Microsoft Research.

24th EACSL Annual Conference on Computer Science Logic (CSL 2015).
Editor: Stephan Kreutzer; pp. 1–1

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Temporal Logics with Local Constraints*

Claudia Carapelle[1] and Markus Lohrey[2]

1 **University of Leipzig, Germany**
carapelle@informatik.uni-leipzig.de
2 **University of Siegen, Germany**
lohrey@eti.uni-siegen.de

Abstract

Recent decidability results on the satisfiability problem for temporal logics, in particular LTL, CTL* and ECTL*, with constraints over external structures like the integers with the order or infinite trees are surveyed in this paper.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases Temporal logics with constraints, concrete domains, LTL, CTL*, ECTL*

Digital Object Identifier 10.4230/LIPIcs.CSL.2015.2

Category Invited Talk

1 Linear Time Temporal Logic with Constraints

Temporal logics are a very popular family of logical languages, used to specify properties of abstracted systems. Pnueli [27] was the first who used linear temporal logic, briefly denoted by LTL, for reasoning about reactive systems. Since then, LTL has become one of the most prominent specification languages used in verification and model checking. Both, model-checking and satisfiability for LTL are PSPACE-complete [28].

In the last few years, many extensions of temporal logics have been proposed in order to address the need to express more than just abstract properties, see for instance [2, 4, 16, 17, 32]. In some of these studies we can find languages which allow to reason about time intervals, space regions, data values from dense domains like the real numbers or discrete domains like the integers or natural numbers.

A general approach for creating such formalisms is described by Demri and D'Souza in [15], where they show how to extend LTL with the ability to express properties of data values from an arbitrary relational structure $\mathcal{D}$, which is often called a *concrete domain*. An example of a concrete domain can be $(\mathbb{Z}, <)$, whose universe is the set $\mathbb{Z}$ of integers and $<$ is the standard linear order on $\mathbb{Z}$, viewed as a binary relation. The approach from [15] is also used in the field of description logics (DLs), where Baader and Hanschke first described a way to integrate arbitrary concrete domains into the knowledge-representation language $\mathcal{ALC}$ [3].

The logic defined in [15] is called constraint LTL, briefly CLTL. The idea behind this language is the following: Fix a set of variables $\mathcal{X}$ and another one of propositions $\mathcal{P}$, both countably infinite, for the rest of the paper. Moreover, fix a relational signature σ, which is a set of relational symbols R, each having an arity a_R. We assume that σ is either finite or countably infinite. A σ-structure is a tuple $\mathcal{D} = (D, (R^{\mathcal{D}})_{R\in\sigma})$, where $R^{\mathcal{D}} \subseteq D^{a_R}$ is a relation

* This work was partially supported by the DFG Research Training Group 1763 (QuantLA).

of arity a_R. In the following, we always identify the relational symbol R with the associated relation $R^{\mathcal{D}}$ if the structure $\mathcal{D}$ is clear from the context. Then the set of CLTL-formulas over $\mathcal{D}$ is defined by the following syntax, where $p \in \mathcal{P}$, $R \in \sigma$, $k = a_R$, $i_1, \ldots, i_k \in \mathbb{N}$, and $x_1, \ldots, x_k \in \mathcal{X}$:

$$\varphi \ ::= \ p \ \mid \ R(\mathsf{X}^{i_1} x_1, \ldots, \mathsf{X}^{i_k} x_k) \ \mid \ \neg\varphi \ \mid \ \varphi \wedge \varphi \ \mid \ \mathsf{X}\varphi \ \mid \ \varphi \mathsf{U} \varphi \tag{1}$$

A formula of the form

$$R(\mathsf{X}^{i_1} x_1, \ldots, \mathsf{X}^{i_k} x_k) \tag{2}$$

is called a $\mathcal{D}$-*constraint*, or simply constraint if $\mathcal{D}$ is clear from the context. We do not assume that the variables $x_1, \ldots, x_k$ are pairwise distinct. A CLTL-formula over $\mathcal{D}$ is interpreted over an infinite word

$$w = (A_0, \eta_0)(A_1, \eta_1)(A_2, \eta_2) \cdots , \tag{3}$$

where for $i \geq 0$, $A_i \subseteq \mathcal{P}$ is a set of propositions and $\eta_i : \mathcal{X} \to D$ assigns a value from D to every variable. One can think of $\mathcal{D}$-registers attached to the system states. Words of the form (3) are also known as *multi-data words*. For $i \geq 0$ we define the suffix $w[i :]$ as the multi-data word $(A_i, \eta_i)(A_{i+1}, \eta_{i+1})(A_{i+2}, \eta_{i+2}) \cdots$.

The satisfaction relation $w \models \varphi$ where $w = (A_0, \eta_0)(A_1, \eta_1)(A_2, \eta_2) \cdots$ is a multi-data word and φ is a CLTL-formulas over $\mathcal{D}$ is inductively defined as follows (all cases except for the case that φ is a constraint of the form (2) are as for ordinary LTL without constraints):

- $w \models p$ iff $p \in A_0$ for $p \in \mathcal{P}$.
- $w \models R(\mathsf{X}^{i_1} x_1, \ldots, \mathsf{X}^{i_k} x_k)$ iff $(\eta_{i_1}(x_1), \ldots, \eta_{i_k}(x_k)) \in R$.
- $w \models \neg\varphi$ iff $w \models \varphi$ does not hold.
- $w \models \varphi_1 \wedge \varphi_2$ iff $w \models \varphi_1$ and $w \models \varphi_2$.
- $w \models \mathsf{X}\varphi$ iff $w[1 :] \models \varphi$.
- $w \models \varphi_1 \mathsf{U} \varphi_2$ iff there is an $i \geq 0$ with $w[i :] \models \varphi_2$ and $w[j :] \models \varphi_1$ for all $0 \leq j \leq i - 1$.

▶ **Example 1.** Take the structure $\mathcal{D} = (\mathbb{Z}, <, =, (=_a)_{a \in \mathbb{Z}})$, where $<$ is the order relation defined above, and $=_a$ is the unary predicate that only holds for a. Instead of $=_a(x)$ we write $x = a$. The CLTL-formula $(x < \mathsf{X}^1 y)\,\mathsf{U}\,(y = 100)$ holds on a multi-data word if and only if there is a position where variable y holds the value 100 and for all previous positions t, the value of x at position t is strictly smaller than the value of y at position $t + 1$.

2 Satisfiability for Linear Time Temporal Logic with Constraints

A CLTL-formula φ over $\mathcal{D}$ is satisfiable if there exists a multi-data word w of the form (3) such that $w \models \varphi$. Of course, if $\mathcal{P}_\varphi \subseteq \mathcal{P}$ (resp. $\mathcal{X}_\varphi \subseteq \mathcal{X}$) is the finite set of propositions (resp., variables) that occur in φ then we can assume that $A_i \subseteq \mathcal{P}_\varphi$ and $\eta_i : \mathcal{X}_\varphi \to D$ in (3).

Balbiani and Condotta [4] proved a general decidability result for CLTL over concrete domains $\mathcal{D}$ satisfying certain properties. The following outline follows [15], where the result of Balbiani and Condotta is reproved in an automata theoretic framework. First of all, let us fix a concrete domain $\mathcal{D} = (D, R_1, \ldots, R_n)$ with only finitely many relations. If $i_j = 0$ for all $1 \leq j \leq k$ in (2), then we call the $\mathcal{D}$-constraint a *point* $\mathcal{D}$-*constraint* (since it refers to one time point). For a point $\mathcal{D}$-constraint $R(x_1, \ldots, x_k)$ and a mapping $\eta : V \to D$, where $x_1, \ldots, x_k \in V \subseteq \mathcal{X}$ we write $\eta \models R(x_1, \ldots, x_k)$ if $(\eta(x_1), \ldots, \eta(x_k)) \in R$. Given a finite subset $V \subseteq \mathcal{X}$ of variables and a mapping $\eta : V \to D$ we denote with $\mathsf{frame}(V, \eta)$ the set of all constraints $R(x_1, \ldots, x_k)$ with $x_1, \ldots, x_k \in V$ and $\eta \models R(x_1, \ldots, x_k)$. A *frame* over

the finite subset $V \subseteq \mathcal{X}$ is a set of constraints of the form $\mathsf{frame}(V, \eta)$ for some mapping $\eta : V \to \mathcal{D}$. In other words, a frame over V is a maximal set of constraints which is still satisfiable. We say that *frame-checking* is decidable for $\mathcal{D}$ if there exists an algorithm, whose input is a finite set of constraints C and which checks, whether C is a frame.

For a set of constraints C and a set of variables $U \subseteq \mathcal{X}$ we denote with $C{\restriction_U} \subseteq C$ the set of all constraints $R(x_1, \dots x_k) \in C$ such that $x_1, \dots, x_k \in U$. A structure $\mathcal{D}$ has the *completion property*, if for every frame C over V, every subset $V' \subseteq V$, and every mapping $\eta' : V' \to D$ such that $C{\restriction_{V'}} = \mathsf{frame}(V', \eta')$ there exists an extension η of η' (meaning that $\eta'(x) = \eta(x)$ for all $x \in V'$) such that $C = \mathsf{frame}(V, \eta)$.

Now we can present the result of Balbiani and Condotta [4] in the form stated in [15]:

▶ **Theorem 2** ([4, 15]). *Let $\mathcal{D} = (D, R_1, \dots, R_n)$ be a structure having the completion property. If frame-checking is decidable (resp., in* PSPACE*), then satisfiability for* CLTL *over $\mathcal{D}$ is decidable (resp.,* PSPACE*-complete).*

Recall that satisfiability for ordinary LTL (without constraints) is already PSPACE-complete. For the proof of Theorem 2 one follows the classical translation of an LTL-formula (without constraints) to a Büchi automaton. In addition to a set of subformulas, the Büchi automaton also has to store a frame over the variables appearing in the CLTL-formula. Along its run, the Büchi automaton checks whether successive frames fit together in the sense that they can be extended to a single frame. Moreover, the automaton checks whether the constraints that belong to the current set of subformulas hold in that combined frame (for this, one has to assume that $i_1, \dots, i_k \leq 1$, which indeed can be enforced by adding further variables). The resulting Büchi automaton accepts a non-empty language if and only if the CLTL-formula is satisfiable. The completion property ensures the correctness of the construction.

Instances of domains with the completion property and decidable frame-checking are $(D, <, =)$ with $D = \mathbb{R}$ or $D = \mathbb{Q}$, and $(\mathbb{R}^2, sw, s, se, w, e, nw, n, ne, =)$, where the nine relations illustrate the mutual position of two points in the Cartesian plane (eg. (a, b) sw (c, d) iff $a < c$ and $b < d$). In these cases, the dense structure of the real and rational numbers is fundamental for the completion property. On the other hand, the structures $(\mathbb{N}, <)$ and $(\mathbb{Z}, <)$ do not have the completion property: Take for instance the frame consisting of the constraints $x < y, y < z, x < z$. Then the mapping $\eta' : \{x, z\} \to \mathbb{N}$ with $\eta'(x) = 1$ and $\eta'(z) = 2$ cannot be extended to a mapping $\eta : \{x, y, z\} \to \mathbb{N}$ such that $\mathsf{frame}(\{x, y, z\}, \eta) = \{x < y, y < z, x < z\}$. In fact, it was shown in [15] that a structure $(D, <, =)$ where $(D, <)$ is an infinite linear order satisfies the completion property if and only if $(D, <)$ is dense and has neither a smallest nor a largest element. This originated the question whether satisfiability of CLTL over $(\mathbb{Z}, <, =)$ or $(\mathbb{N}, <, =)$ is still decidable. Demri and D'Souza [15] finally answered this question positively (as in Example 1, $=_a$ denotes the unary relation $\{a\}$):

▶ **Theorem 3** ([15]). *Satisfiability for* CLTL *over the structures $(\mathbb{Z}, <, =, (=_a)_{a \in \mathbb{Z}})$ and $(\mathbb{N}, <, =, (=_a)_{a \in \mathbb{N}})$ is* PSPACE*-complete.*

In [16], Demri and Gascon extend this result to CLTL with so called IPC*-constraints. If we disregard succinctness aspects, this logic is equivalent to CLTL over the structure

$$\mathcal{Z} = (\mathbb{Z}, <, =, (=_a)_{a \in \mathbb{Z}}, (\equiv_{a,b})_{0 \leq a < b}), \tag{4}$$

where $\equiv_{a,b}$ denotes the unary relation $\{a + xb \mid x \in \mathbb{Z}\}$ (expressing that an integer is congruent to a modulo b). The main result from [16] states that satisfiability of CLTL with IPC*-constraints is still PSPACE-complete.

Constraints over the above structure (4) do not allow to express the successor relation $y = x + 1$, which would be very useful for analyzing counter systems. There is a good reason for this: Using successor constraints it is easy to reduce the halting problem (and even the Σ_1^1-complete recurrent reachability problem) for two-counter machines to the satisfiability problem for CLTL over $(\mathbb{Z}, \{(x, x+1) \mid x \in \mathbb{Z}\})$. In [15] the authors extend this observation by showing undecidability of satisfiability for CLTL over every structure with a so called *implicit counting mechanism*. A relational structure $\mathcal{D}$ with universe D has an implicit counting mechanism if it contains the equality relation and a binary relation R such that (i) $R = \{(x, y) \in D \times D \mid f(x) = y\}$, where $f : D \to D$ is injective and (ii) (D, R) is acyclic.

▶ **Theorem 4** ([15]). *If $\mathcal{D}$ has an implicit counting mechanism, then satisfiability for* CLTL *over $\mathcal{D}$ is hard for Σ_1^1 (the first existential level of the analytical hierarchy).*

Using the structure $\mathcal{Z}$ from (4) one can still specify an abstracted version of increment operations. For example $x = y+1$ can be abstracted by $(y > x) \wedge \bigvee_{i=-2^k}^{2^k-1} (\equiv_{i,2^k}(x) \wedge \equiv_{i+1,2^k}(y))$ where k is a large natural number. This is why CLTL over $\mathcal{Z}$ seems to be a good compromise between (unexpressive) total abstraction and (undecidable) high concretion.

3 Branching Time Temporal Logics with Constraints

In the same way as outlined for LTL in Section 1, constraints can be also added to branching time logics like CTL* and even ECTL* (extended computation tree logic), obtaining CCTL* and CECTL*, respectively. Formulas from these logics are interpreted over *decorated* Kripke structures. Fix again a σ-structure $\mathcal{D} = (D, (R^{\mathcal{D}})_{R \in \sigma})$. A $\mathcal{D}$-decorated Kripke structure is a tuple $\mathcal{K} = (V, R, \lambda, \zeta)$, where V is the set of nodes (or states), $R \subseteq V \times V$ is a binary edge relation such that for every $v \in V$ there exists $v' \in V$ with $(v, v') \in R$, and for every node $v \in V$, $\lambda(v) \subseteq \mathcal{P}$ is the set of propositions that hold in v, whereas $\zeta(v) : \mathcal{X} \to D$ assigns values from D to the variables (or registers). Instead of $(\zeta(v))(x)$ we also write $\zeta(v, x)$. For $v \in V$, a $(\mathcal{K}, v)$-path is an infinite sequence of nodes $\rho = (v_0, v_1, v_2, \ldots)$ such that $v_0 = v$ and $(v_i, v_{i+1}) \in R$ for $i \geq 0$. As for multi-data words we define $\rho[i :] = (v_i, v_{i+1}, v_{i+2}, \ldots)$.

The syntax of CCTL* over $\mathcal{D}$ is given by the following grammar, where $p \in \mathcal{P}$, $R \in \sigma$, $k = a_R$ is the arity of R, $i_1, \ldots, i_k \in \mathbb{N}$, and $x_1, \ldots, x_k \in \mathcal{X}$:

$$\varphi \quad ::= \quad p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathsf{E}\psi \tag{5}$$

$$\psi \quad ::= \quad \varphi \mid R(\mathsf{X}^{i_1} x_1, \ldots, \mathsf{X}^{i_k} x_k) \mid \neg\psi \mid \psi \wedge \psi \mid \mathsf{X}\psi \mid \psi \mathsf{U} \psi \tag{6}$$

Formulas of the form (5) are state formulas and are interpreted over nodes of a $\mathcal{D}$-decorated Kripke structure, whereas formulas of the form (6) are path formulas and are interpreted over paths in $\mathcal{K}$. Note that constraints are path formulas. Here is the inductive definition of the satsifiability relation, where $\mathcal{K} = (V, R, \lambda, \zeta)$ is a $\mathcal{D}$-decorated Kripke structure, $v \in V$ and $\rho = (v_0, v_1, v_2, \ldots)$ is a $(\mathcal{K}, v_0)$-path (we omit here the obvious cases for the boolean operators $\neg$ and $\wedge$):

- $(\mathcal{K}, v) \models p$ iff $p \in \lambda(v)$ for $p \in \mathcal{P}$.
- $(\mathcal{K}, v) \models \mathsf{E}\psi$ iff there is a $(\mathcal{K}, v)$-path ϱ such that $(\mathcal{K}, \varrho) \models \psi$.
- $(\mathcal{K}, \rho) \models \varphi$ if $(\mathcal{K}, v_0) \models \varphi$ for a state formula φ.
- $(\mathcal{K}, \rho) \models R(\mathsf{X}^{i_1} x_1, \ldots, \mathsf{X}^{i_k} x_k)$ if $(\zeta(v_{i_1}, x_1), \ldots, \zeta(v_{i_k}, x_k)) \in R$.
- $(\mathcal{K}, \rho) \models \mathsf{X}\varphi$ iff $(\mathcal{K}, \rho[1 :]) \models \varphi$.
- $(\mathcal{K}, \rho) \models \varphi_1 \mathsf{U} \varphi_2$ iff there is an $i \geq 0$ with $(\mathcal{K}, \rho[i :]) \models \varphi_2$ and $(\mathcal{K}, \rho[j :]) \models \varphi_1$ for all $0 \leq j \leq i - 1$.

A CCTL^* state formula ψ over $\mathcal{D}$ is satisfiable if there exists a $\mathcal{D}$-decorated Kripke structure $\mathcal{K}$ and a node v from $\mathcal{D}$ such that $(\mathcal{K}, v) \models \psi$.

A weak form of CCTL^* over $\mathcal{Z}$, where only integer variables at the same state can be compared, was first introduced in [13] and used to describe properties of so called relational automata. It was shown in [13] that the model checking problem for the above fragment of CCTL^* over relational automata is undecidable.

Demri and Gascon [16] asked whether satisfiability of CCTL^* over the structure $\mathcal{Z}$ from (4) is decidable. This problem was further investigated in [6, 20], where several partial results where shown: If we replace in $\mathcal{Z}$ the binary predicate $<$ by unary predicates $<_c = \{x \mid x < c\}$ for $c \in \mathbb{Z}$, then satisfiability of CCTL^* was shown to be decidable in [20]. For the full structure $\mathcal{Z}$ satisfiability was shown to be decidable for CEF^+, a fragment of CCTL^* which contains both the existential and the universal fragment of CCTL^*, see [6] for details. Later in [7] Bozzelli and Pinchinat proved that satisfiability of the existential and universal fragment of CCTL^* over the domain $(\mathbb{Z}, =, <)$ are PSPACE-complete. Finally, in [11], we answered the question of Demri and Gascon positively:

▶ **Theorem 5** ([11]). CCTL^* *over* $\mathcal{Z}$ *is decidable.*

Before we explain the proof techniques from [11], let us first discuss an extension of Therem 5 to ECTL^* (extended CTL^*) with constraints over $\mathcal{Z}$, which was shown in the long version [12] of [11] using a straightforward extension of the techniques from [11].

The logic ECTL^* (without constraints) is a proper extension of CTL^* (see [29, 31]) in which path formulas are defined by Büchi-automata or, equivalently, MSO-formulas. In contrast, CTL^* can only specify LTL-properties or, equivalently, first-order properties along paths. To define the constraint version of ECTL^* over $\mathcal{D} = (D, (R^{\mathcal{D}})_{R \in \sigma})$ we first have to define a constraint version of MSO (monadic second-order logic) over infinite words, which is interpreted over multi-data words. We speak of CMSO (constraint MSO) over $\mathcal{D}$. Fix a countably infinite set $\mathcal{V}_{\mathsf{el}}$ (resp., $\mathcal{V}_{\mathsf{set}}$) of element variables (resp., set variables). The set of CMSO-formulas over $\mathcal{D}$ is defined by the following grammar, where $y, y_1, y_2 \in \mathcal{V}_{\mathsf{el}}$, $Y \in \mathcal{V}_{\mathsf{set}}$, $p \in \mathcal{P}$, $R \in \sigma$, $k = a_R$ is the arity of R, $i_1, \ldots, i_k \in \mathbb{N}$, and $x_1, \ldots, x_k \in \mathcal{X}$:

$$\varphi \ ::= \ p(y) \mid y_1 < y_2 \mid y \in Y \mid [R(\mathsf{X}^{i_1} x_1, \ldots, \mathsf{X}^{i_k} x_k)](y) \mid \neg\varphi \mid \varphi \wedge \varphi \mid \exists y\, \varphi \mid \exists Y\, \varphi$$

To define the semantics of CMSO over $\mathcal{D}$ we need interpretation functions $I_1 : \mathcal{V}_{\mathsf{el}} \to \mathbb{N}$ and $I_2 : \mathcal{V}_{\mathsf{set}} \to 2^{\mathbb{N}}$. Then, for a multi-data word $w = (A_0, \eta_0)(A_1, \eta_1)(A_2, \eta_2) \cdots$ we define $(w, I_1, I_2) \models \varphi$ inductively as follows (again we omit the obvious cases for boolean operators):

- $(w, I_1, I_2) \models p(y)$ iff $p \in A_j$ where $j = I_1(y)$.
- $(w, I_1, I_2) \models y_1 < y_2$ iff $I_1(y_1) < I_2(y_2)$.
- $(w, I_1, I_2) \models y \in Y$ iff $I_1(y) \in I_2(Y)$.
- $(w, I_1, I_2) \models [R(\mathsf{X}^{i_1} x_1, \ldots, \mathsf{X}^{i_k} x_k)](y)$ iff $(\eta_{j+i_1}(x_1), \ldots, \eta_{j+i_k}(x_k)) \in R$, where $j = I_1(y)$.
- $(w, I_1, I_2) \models \exists y\, \varphi$ iff there exists $j \in \mathbb{N}$ such that $(w, I_1[y \mapsto j], I_2) \models \varphi$.
- $(w, I_1, I_2) \models \exists y\, \varphi$ iff there exists $J \subseteq \mathbb{N}$ such that $(w, I_1, I_2[Y \mapsto J]) \models \varphi$.

Here, the function $I_1[y \mapsto j]$ is defined by $I_1[y \mapsto j](y) = j$ and $I_1[y \mapsto j](y') = I_1(y')$ for $y' \neq y$, and similarly for $I_2[Y \mapsto J]$. Moreover, for a pair $(\mathcal{K}, \rho)$ consisting of a $\mathcal{D}$-decorated Kripke structure $\mathcal{K} = (D, R, \lambda, \zeta)$ and a $(\mathcal{K}, v_0)$-path $\rho = (v_0, v_1, v_2, \ldots)$ (for some node v_0 of $\mathcal{K}$) we write $(\mathcal{K}, \rho, I_1, I_2) \models \varphi$ if $(w, I_1, I_2) \models \varphi$, where w is the multi-data word $(\lambda(v_0), \zeta(v_0))(\lambda(v_1), \zeta(v_1))(\lambda(v_2), \zeta(v_2)) \cdots$.

Now we can define CECTL^* (constraint ECTL^*) over $\mathcal{D} = (D, (R^{\mathcal{D}})_{R \in \sigma})$ as follows, where φ is an arbitrary CMSO-formula over $\mathcal{D}$ in which only the set variables $Y_1, \ldots, Y_n$ occur freely:

$$\varphi \ ::= \ \neg\varphi \mid \varphi \wedge \varphi \mid \mathsf{E}\varphi[Y_1/\varphi, \ldots, Y_n/\varphi]$$

Such a formula is evaluated in a node $v \in D$ of a $\mathcal{D}$-decorated Kripke structure $\mathcal{K} = (V, R, \lambda, \zeta)$ by the following rule (the definition for boolean operator is the obvious one): $(\mathcal{K}, v_0) \models \mathsf{E}\varphi[Y_1/\varphi_1, \ldots, Y_n/\varphi_n]$ iff there is a $(\mathcal{K}, v_0)$-path $\rho = (v_0, v_1, v_2, \ldots)$ such that $(\mathcal{K}, \rho, I_1, I_2) \models \varphi$, where I_1 is arbitrary (note that φ is not allowed to have free element variables) and I_2 satisfies $I_2(Y_i) = \{j \mid (\mathcal{K}, v_j) \models \varphi_i\}$. Note that for a CMSO-formula φ without free variables, $\mathsf{E}\varphi$ is a CECTL* formula that holds in a node v if there is a path starting in v along which φ holds. Satisfiability for CECTL*-formulas over $\mathcal{D}$ is defined as for CCTL*. The main result of [12] is:

▶ **Theorem 6** ([12]). *Satisfiability for* CECTL* *over* $\mathcal{Z}$ *is decidable.*

In the next section, we explain the method that we use to obtain the results from [11, 12], which we call the EHD-*method*.

4 EHD-method

The EHD-method yields sufficient conditions on a relational structure $\mathcal{D}$ which guarantee that satisfiability of CECTL* over $\mathcal{D}$ is decidable. Then, one can show that the structure $\mathcal{Z}$ satisfies these properties.

The structure $\mathcal{D} = (D, (R^{\mathcal{D}})_{R\in\sigma})$ is *negation closed*, if for every $R \in \sigma$ the complement of $R^{\mathcal{D}}$ is definable in positive existential first-order logic over $\mathcal{D}$. Moreover, since σ can be countably infinite we have to require that a positive existential first-order formula for the complement of $R^{\mathcal{D}}$ is computable from the relational symbol $R \in \sigma$. For instance $(\mathbb{Z}, =, <)$ is negation closed, because $\neg x < y$ iff $(x = y \ \vee \ y < x)$ and $\neg x = y$ iff $(x < y \ \vee \ y < x)$. Negation closure is needed in order to achieve a strong kind of negation normal form for CECTL*, in which the constraints only appear positively.

The second condition on $\mathcal{D}$, the EHD-*property*, expresses the fact that we can provide a characterization of all structures which allow a homomorphism into $\mathcal{D}$ using a suitable logical language. Let $\tau \subseteq \sigma$ be a subsignature of σ. A homomorphism $h : \mathcal{C} \to \mathcal{D}$ from a τ-structure $\mathcal{C} = (C, (R^{\mathcal{C}})_{R\in\tau})$ to the σ-structure $\mathcal{D}$ is a mapping $h : C \to D$ such that for every $R \in \tau$ and every tuple $(c_1, \ldots, c_k) \in R^{\mathcal{C}}$ (where $k = a_R$ is the arity of R) we have $(h(c_1), \ldots, h(c_k)) \in R^{\mathcal{D}}$. We say that the σ-structure $\mathcal{D}$ has the property EHD($\mathcal{L}$) for some logic $\mathcal{L}$ if and only if there is a computable function that maps a finite subsignature $\tau \subseteq \sigma$ to an $\mathcal{L}$-sentence φ_τ such that for any countable τ-structure $\mathcal{C}$ one has:

$$\exists h : \mathcal{C} \to \mathcal{D} \text{ homomorphism } \iff \mathcal{C} \models \varphi_\tau .$$

To make use of this condition for proving satisfiability of CECTL*, the logic $\mathcal{L}$ has to satisfy two properties: (i) it has to be at least as expressive as MSO and (ii) the satisfiability problem over the class of infinite node labelled rooted trees has to be decidable for $\mathcal{L}$. In [11, 12] we used for $\mathcal{L}$ the logic Bool(MSO, WMSO+B) (in short BMWB), whose formulas are all boolean combinations of MSO and WMSO+B formulas. Here, WMSO+B is the extension of weak monadic second-order logic (where only quantification over finite subsets is allowed) with the bounding quantifier B: A formula BX φ holds in a structure $\mathcal{A}$ if and only if there exists a bound $b \in \mathbb{N}$ such that for every finite subset B of the domain of $\mathcal{A}$ with $\mathcal{A} \models \varphi(B)$ we have $|B| \leq b$. Recently, Bojańczyk and Toruńczyk have shown that satisfiability of WMSO+B over infinite node-labeled trees is decidable [5]. They translate WMSO+B-formulas into a certain kind of tree automata, which they call puzzles. Since puzzles are equipped with a parity acceptance condition, it follows easily from [5] that satisfiability over infinite node labelled trees remains decidable for BMWB. The technical main result from [12] is:

▶ **Theorem 7** ([12]). *Let $\mathcal{D}$ be a relational structure which is (i) negation closed and (ii) has the property* EHD(BMWB). *Then satisfiability of* CECTL* *over $\mathcal{D}$ is decidable.*

Let us sketch the proof of this result for CCTL*, which is notationally a bit simpler than the proof for CECTL*. So, let φ be a CCTL* state formula. Using negation closure, we can assume that φ is in a strong negation normal form where negations only appear directly in front of atomic propositions $p \in \mathcal{P}$. For this we have to add dual operators to the logic (e.g., the universal path quantifier A). Negation closure allows to eliminate a negation in front of a constraint. Let r be 1+ the number of subformulas of φ of the form $\mathsf{E}\theta$.

Next, it is easy to show that φ is satisfiable if and only if it has a model $\mathcal{T} = (V, R, \lambda, \zeta)$, where (V, R) is a rooted tree of degree r, meaning that $(\mathcal{T}, v_0) \models \varphi$, where v_0 is the root of (V, R). We call such a structure a $\mathcal{D}$-decorated r-tree. The proof for this tree model property is the same as for classical CTL*.

We use the following notation for ancestors in a tree: Let (V, R) be a rooted tree and let $v \in V$. Then we denote with v^1 the parent node of v if it exists. Moreover, for $i \geq 0$ let $v^0 = v$ and $v^{i+1} = (v^i)^1$ (the latter does not necessarily exist). So, v^i is the i-th ancestor of v if it exists.

Next we define an abstracted version of φ, where every occurrence of a constraint $\theta = R(\mathsf{X}^{i_1}x_1, \ldots, \mathsf{X}^{i_k}x_k)$ is replaced by $\mathsf{X}^d p_\theta$, where p_θ is a fresh proposition associated with θ. Here $d = \max\{i_1, \ldots, i_k\}$ is the *depth* of the constraint. We call the resulting formula φ^a; it is a pure CTL*-formula. For a $\mathcal{D}$-decorated r-tree $\mathcal{T} = (V, R, \lambda, \zeta)$ we also define an abstracted version $\mathcal{T}^a = (V, R, \lambda^a)$ (we call it an undecorated r-tree), which is obtained from $\mathcal{T}$ by removing the decoration mapping ζ and adding propositions. More precisely, the labelling function $\lambda^a : V \to 2^{\mathcal{P}}$ is defined as follows: For every node $v \in V$, $\lambda^a(v)$ is the union of $\lambda(v)$ and the set of all fresh propositions p_θ, where θ is a constraint in φ of depth d and θ holds in the path starting in v^d and passing through v. Since θ looks only d steps into the future, it does not matter how the chosen path continues from v downwards in the tree; only the initial segment from the d-th ancestor of v to v is relevant. Note that if the $\mathcal{D}$-decorated r-tree $\mathcal{T}$ is a model for φ, then $\mathcal{T}^a$ is a model for φ^a, but the converse does in general not hold. In order to get an equivalence, we have to add a further condition.

Let $\mathcal{P}_0$ be the set of propositions that appear in the abstracted formula φ^a (which contains the fresh propositions p_θ for constraints θ) and let $\mathcal{X}_0 \subseteq \mathcal{X}$ be the finite set of variables that occur in the initial CCTL*-formula φ. Assume now that $\mathcal{S} = (V, R, \lambda)$ is an undecorated r-tree, where the propositions from $\mathcal{P}_0$ occur. We define a σ-structure $\mathcal{C}_\mathcal{S}$ as follows (σ is the signature of $\mathcal{D}$): Its universe is the set of all pairs $(v, x) \in V \times \mathcal{X}_0$. Moreover, for every k-ary relation symbol $R \in \sigma$ let $R^{\mathcal{C}_\mathcal{S}}$ (the interpretation of R in the structure $\mathcal{C}_\mathcal{S}$) consist of all tuples $((v^{d-i_1}, x_1), \ldots, (v^{d-i_k}, x_k))$ such that $v \in V$, $\lambda(v)$ contains the proposition θ, θ is the constraint $R(\mathsf{X}^{i_1}(x_1), \ldots, \mathsf{X}^{i_k}(x_k))$, and d is the depth of the constraint. The intuition here is the following: The universe of $\mathcal{C}_\mathcal{S}$ is obtained by attaching to each node of $\mathcal{S}$ copies of the variables from $\mathcal{X}_0$. That a node v is labelled with the proposition p_θ indicates that in the unabstracted version of $\mathcal{S}$ (imagine $\mathcal{S}$ is the abstracted version $\mathcal{T}^a$ of a $\mathcal{D}$-decorated r-tree $\mathcal{T}$) the constraint θ holds in the path starting in v^d and passing through v. The relation $R^{\mathcal{C}_\mathcal{S}}$ contains therefore all tuples that are forced to exist due to the labels p_θ for the constraints θ.

With the above definitions, it is not difficult to prove that the following two statements are equivalent (here, we actually need the fact that constraints only occur positively in φ):

- There is a $\mathcal{D}$-decorated r-tree $\mathcal{T}$ with root v_0 such that $(\mathcal{T}, v_0) \models \varphi$
- There is an undecorated r-tree $\mathcal{S}$ with root v_0 and a homomorphism $h : \mathcal{C}_\mathcal{S} \to \mathcal{D}$ such that $(\mathcal{S}, v_0) \models \varphi^a$.

Now it is fairly easy to finish the proof of Theorem 7. By the EHD(BMWB)-property, there

exists a BMWB-sentence ψ such that $\mathcal{C}_{\mathcal{S}} \models \psi$ if and only if there is a homomorphism $h : \mathcal{C}_{\mathcal{S}} \to \mathcal{D}$. From the definition of the structure $\mathcal{C}_{\mathcal{S}}$ it is easy to see that it can be obtained by a so called copying first-order transduction from $\mathcal{S}$. One can therefore construct from the BMWB-sentence ψ another BMWB-sentence ψ' such that $\mathcal{C}_{\mathcal{S}} \models \psi$ if and only if $\mathcal{S} \models \psi'$. Finally, since CTL* can be translated into MSO, one can construct an MSO sentence ψ'' such that $(\mathcal{S}, v_0) \models \varphi^a$ if and only if $\mathcal{S} \models \psi''$. Altogether we have that our initial CCTL*-formula φ is satisfiable if and only if there exists an r-tree $\mathcal{S}$ such that $\mathcal{S} \models \psi' \wedge \psi''$. By the result of Bojańczyk and Toruńczyk [5] the latter is decidable. This concludes our proof sketch for Theorem 7.

By Theorem 7, to prove Theorem 6 it suffices to show that the structure $\mathcal{Z}$ from (4) is negation closed and has the property EHD(BMWB). Negation closure is straightforward to show. For the EHD(BMWB)-property let us briefly argue why the reduct $(\mathbb{Z}, <)$ of $\mathcal{Z}$ has the property EHD(BMWB). For a structure $\mathcal{C} = (C, R)$ (where R is an arbitrary binary relation on C) one can show that there exists a homomorphism from $\mathcal{C}$ to $(\mathbb{Z}, <)$ if and only if (i) R is acyclic and (ii) for all $a, b \in C$ there is a bound on the length of all paths in $\mathcal{C}$ from a to b. These properties can be easily expressed in WMSO+B, which shows that $(\mathbb{Z}, <)$ has the property EHD(BMWB). Moreover, the above characterization of the existence of homomorphisms to $(\mathbb{Z}, <)$ can be extended to $\mathcal{Z}$. Let us only remark that one has to consider also structures $\mathcal{C}$ (over the same signature as $\mathcal{Z}$), where the equality symbol $=$ is not interpreted by the identity relation on the universe of $\mathcal{C}$.

Our proof that $\mathcal{Z}$ has the property EHD(BMWB) actually only needs rather weak assumptions on the unary predicates (which are satisfied for the unary relations $=_a$ and $\equiv_{a,b}$). In particular, Theorem 6 can be extended to expansions of $\mathcal{Z}$ that contain additional unary predicates like the set of primes and even some undecidable subsets of $\mathbb{Z}$, see [12] for details.

The EHD-method is quite general, and it is tempting to try applying it to other structures. An interesting candidate in this context (as mentioned in [16]) is the infinite order tree

$$\mathcal{T}_\infty = (\mathbb{N}^*, <, =, \perp),$$

where $<$ denotes the prefix order on $\mathbb{N}^*$ and $\perp$ denotes the incomparability relation with respect to $<$. We add the latter relation in order to obtain a negation closed structure. Unfortunately, using an Ehrenfeucht-Fraïssé-game for WMSO+B, we proved in [10] that $\mathcal{T}_\infty$ does not satisfy the property EHD(BMWB):

▶ **Theorem 8** ([10]). *There is no* BMWB*-sentence ψ such that for every countable structure $\mathcal{A}$ over the signature $\{<, =, \perp\}$ one has: $\mathcal{A} \models \psi$ if and only if there is a homomorphism $h : \mathcal{A} \to \mathcal{T}_\infty$.*

In other words, BMWB is not expressive enough to distinguish between those $\{<, =, \perp\}$-structures which can be mapped homomorphically to the infinite order tree and those that cannot.

This shows that the EHD-method cannot be applied to the concrete domain $\mathcal{T}_\infty$ (or, equivalently, to the infinite binary tree), but it does not imply that satisfiability for CECTL* over $\mathcal{T}_\infty$ is undecidable. In fact, recently Demri and Deters [14] gave a positive answer for CCTL* over $\mathcal{T}_\infty$:

▶ **Theorem 9** ([14]). *Satisfiability for* CCTL* *over $\mathcal{T}_\infty$ is decidable and* PSPACE*-completeness for the corresponding* CLTL*-fragment.*

The result for CLTL has been recently reproved in [21] using a direct automata theoretic approach. Demri and Deters prove their results actually for a richer logic, which allows to

compare the length of the longest common prefix for pairs of elements from $\mathcal{T}_\infty$. Decidability is obtained by a reduction to the satisfiability problem of CLTL (resp., CCTL*) over the domain $(\mathbb{N}, =, <, (=_a)_{a\in\mathbb{N}})$, which is PSPACE-complete (resp., decidable) by [16] (resp., [11]). We conjecture that the decidability result for CCTL* over $\mathcal{T}_\infty$ can be extended to CECTL*.

Despite the fact that the EHD-method fails for $\mathcal{T}_\infty$, one can apply it to other tree-like structures, such as semi-linear orders, ordinal trees, and infinitely branching trees of a fixed height. *Semi-linear orders* are partial orders that are tree-like in the sense that for every element x the set of all smaller elements $\downarrow x$ forms a linear suborder. If this linear suborder $\downarrow x$ is an ordinal (for every x) then one has an *ordinal tree*. Ordinal trees are widely studied in descriptive set theory and recursion theory. Note that a tree is a particular instance of a semi-linear order which has a smallest element and where for every x the set $\downarrow x$ is finite.

So far, we have investigated satisfiability for CECTL* over one fixed structure $\mathcal{D}$. For semi-linear orders and ordinal trees it is more natural to consider satisfiability with respect to a class of concrete domains Γ (over a fixed signature σ): The question becomes, whether for a given CECTL*-formula φ there is a concrete domain $\mathcal{D} \in \Gamma$ such that φ holds in a $\mathcal{D}$-decorated Kripke structure. If a class Γ has a universal structure[1] $\mathcal{U}$, then satisfiability with respect to the class Γ is equivalent to satisfiability with respect to $\mathcal{U}$ because there is a $\mathcal{D} \in \Gamma$ such that φ holds in a $\mathcal{D}$-decorated Kripke structure if and only if φ holds in a $\mathcal{U}$-decorated Kripke structure. A typical class with a universal structure is the class of all countable linear orders, for which $(\mathbb{Q}, <)$ is universal. Similarly, for the class of all countable trees the tree $\mathcal{T}_\infty$ as well as the infinite binary tree are universal.

Using the EHD-method, we proved the following decidability results in [10]:

▶ **Theorem 10** ([10]). *Satisfiability of* CECTL* *over each of the following classes is decidable:*

- *the class of all semi-linear orders,*
- *the class of all ordinal trees, and*
- *for each $h \in \mathbb{N}$, the class of all order trees of height h.*

5 Adding Non-Local Constraints

Notice that the constraints of the form $R(\mathsf{X}^{i_1}x_1, \ldots, \mathsf{X}^{i_k}x_k)$ which we have considered so far are *local*, in the sense that they can compare data values in an n-sized neighborhood of the state in which they are evaluated, where $n = \max\{i_1, \ldots, i_k\} + 1$. Other proposed extensions of temporal logics have the ability to compare data values at arbitrary distance. Metric temporal logic (MTL) [2] and FreezeLTL [17] are two prominent examples of such logics. In [15], Demri and D'Souza ask whether satisfiability of CLTL with constraints over the integers is preserved when adding non-local constraints of the form $x = \mathsf{F}y$, stating that there exists a future state where the value of y is equal to the current value of x. Using a reduction from the Π^0_1-complete problem of deciding the existence of an infinite accepting run of an incrementing counter automaton (see [17]), we answered this question negatively in [12]:

▶ **Theorem 11** ([12]). *Satisfiability for* CLTL *over $(\mathbb{Z}, =, <)$ extended with non-local constraints of the form $x = \mathsf{F}y$ is undecidable.*

On the other hand, if one adds non-local constraints of the form $x < \mathsf{F}y$ and $\mathsf{F}x < y$ to CLTL, then one still gets a decidable logic:

[1] A structure $\mathcal{U}$ is universal for the class Γ if (i) $\mathcal{U} \in \Gamma$ and (ii) every structure from Γ is an induced substructure of $\mathcal{U}$.

▶ **Theorem 12.** *Satisfiability for* CLTL *over* $\mathcal{Z}$ *from* (4) *extended with non-local order constraints of the form* $x < \mathsf{F}y$ *and* $\mathsf{F}x < y$ *is* PSPACE*-complete.*

This result was shown in the PhD thesis of the first author [8], where it is shown that non-local order constraints of the form $x < \mathsf{F}y$ and $\mathsf{F}x < y$ can be replaced by local order constraints. It remains open, whether CCTL* over $\mathcal{Z}$ (or the reduct $(\mathbb{Z}, =, <)$) extended with non-local constraints of the form $x < \mathsf{F}y$ and $\mathsf{F}x < y$ is still decidable.

6 Related Work

In the area of knowledge representation, extensions of description logics with constraints on different concrete domains have been intensively studied, see [23] for a survey. In [24], it was shown that the extension of the description logic $\mathcal{ALC}$ with constraints from $(\mathbb{Q}, <, =)$ has a decidable (EXPTIME-complete) satisfiability problem even in the presence of general TBoxes. A TBox can be seen as a second $\mathcal{ALC}$-formula that has to hold in all nodes of a model. Our decidability proof is partly inspired by the construction from [24], which in contrast to our proof is purely automata-theoretic. Further results for description logics and concrete domains can be found in [25, 26].

There are other extensions of temporal logics that allow to reason about structures with data values, especially in the linear time setting. Logical languages like MTL [22, 2] and TPTL [1] are extensions of LTL often used to specify properties of *timed words*, i.e. data words over the real numbers in which the data sequence is monotonically growing, or monotonic data words over the natural numbers. These logics have however also received some attention on non-monotonic data words [9, 19]. In general, as soon as one drops the monotonicity requirement, satisfiability for these logics becomes undecidable and research has been concentrating on some decidable fragments. An example is freezeLTL, a syntactical restriction of TPTL that has the ability to check data values only for equality. Satisfiability for freezeLTL has been shown to be decidable over finite data words, but undecidable over infinite data words [17]. In contrast to CLTL, the constraints of freezeLTL are of a global nature.

7 Open Problems

The most important open problem that remains from this work concerns the complexity of satisfiability for CCTL* over $(\mathbb{Z}, <, =)$ (or even $\mathcal{Z}$). Clearly, this problem is 2EXPTIME-hard, since satisfiability of (unconstrained) CTL* is 2EXPTIME-complete [18, 30]. To get an upper complexity bound, one should investigate the complexity of the emptiness problem for puzzles (the tree automata used in [5] to show that satisfiability of WMSO+B over infinite node labelled trees is decidable). The WMSO+B-properties used in our decidability result for the structure $(\mathbb{Z}, <, =)$ are very simple, in particular there quantifier nesting depth is small. One may hope to derive a reasonable complexity bound from this observation. At the same time, we believe that our decidability result based on the EHD-method, whose upside is its general nature, may not be the most effective way to devise an efficient decidability procedure for the specific case of the structure $(\mathbb{Z}, <, =)$. The reason behind this statement is the following: Recall from our proof sketch for Theorem 7 that we have to check whether there exists a tree $\mathcal{S}$ which (i) is a model of the abstracted CTL*-formula φ^a and (ii) such that there is a homomorphism from the structure $\mathcal{C}_\mathcal{S}$ to $\mathcal{D}$. Property (i) can be checked in 2EXPTIME. The complexity theoretic bottleneck in our proof is property (ii). We simply translate it to BMWB over trees, for which we have no complexity bound. But $\mathcal{C}_\mathcal{S}$ has some interesting properties:

It has bounded degree and bounded tree-width, where the tree-width is determined by the number of variables occurring in the input formula. Maybe one can exploit this fact to come up with a more efficient solution. Let us remark that for CECTL^* satisfiability over any concrete domain is non-elementary since path properties are specified in MSO (for which satisfiability is non-elementary). Of course one may replace MSO by Büchi automata (as in [29, 31]), which might then lead to an elementary complexity bound.

Another interesting question is whether there exists a linear order $(A, <)$ such that satisfiability for CECTL^* (or even CCTL^* or CLTL) over $(A, <, =)$ is undecidable. Such a linear order must be necessarily scattered, i.e., $(\mathbb{Q}, <)$ cannot be embedded into $(A, <)$. If $(\mathbb{Q}, <)$ can be embedded into $(A, <)$, then satisfiability over $(A, <, =)$ is equivalent to satisfiability over $(\mathbb{Q}, <, =)$. Finally, it would be interesting to know, whether there exists a concrete domain $\mathcal{D}$ such that satisfiability for CLTL over $\mathcal{D}$ is decidable but the more expressive CCTL^* over $\mathcal{D}$ is undecidable.

References

1 Rajeev Alur and Thomas A. Henzinger. A really temporal logic. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science (FOCS 1989)*, pages 164–169. IEEE Computer Society Press, 1989.

2 Rajeev Alur and Thomas A. Henzinger. Real-time logics: complexity and expressiveness. *Information and Computation*, 104:390–401, 1993.

3 Franz Baader and Philipp Hanschke. A scheme for integrating concrete domains into concept languages. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI 1991)*, volume 1, pages 452–457. Morgan Kaufmann, 1991.

4 Philippe Balbiani and Jean-François Condotta. Computational complexity of propositional linear temporal logics based on qualitative spatial or temporal reasoning. In *Proceedings of the 4th International Workshop on Frontiers of Combining Systems (FroCos 2002)*, volume 2309 of *Lecture Notes in Computer Science*, pages 162–176. Springer, 2002.

5 M. Bojańczyk and S. Toruńczyk. Weak MSO+U over infinite trees. In *Proceedings of the 29th International Symposium on Theoretical Aspects of Computer Science (STACS 2012)*, volume 14 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 648–660. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2012.

6 Laura Bozzelli and Régis Gascon. Branching-time temporal logic extended with qualitative presburger constraints. In *Proceedings of the 13th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR 2006)*, volume 4246 of *Lecture Notes in Computer Science*, pages 197–211. Springer, 2006.

7 Laura Bozzelli and Sophie Pinchinat. Verification of gap-order constraint abstractions of counter systems. *Theoretical Computer Science*, 523:1–36, 2014.

8 Claudia Carapelle. *On the Satisfiability of Temporal Logics with Concrete Domains*. PhD thesis, University of Leipzig, 2015. submitted.

9 Claudia Carapelle, Shiguang Feng, Oliver Fernández Gil, and Karin Quaas. Satisfiability for MTL and TPTL over Non-monotonic Data Words. In *Proceedings of the 8th International Conference on Language and Automata Theory and Applications (LATA 2014)*, volume 8370 of *Lecture Notes in Computer Science*. Springer, 2014.

10 Claudia Carapelle, Shiguang Feng, Alexander Kartzow, and Markus Lohrey. Satisfiability of ECTL* with Tree Constraints. In *Proceedings of the 10th International Computer Science Symposium in Russia (CSR 2015)*, volume 9139 of *Lecture Notes in Computer Science*, pages 94–108. Springer, 2015.

11 Claudia Carapelle, Alexander Kartzow, and Markus Lohrey. Satisfiability of CTL* with Constraints. In *Proceedings of the 24th International Conference on Concurrency The-*

ory (CONCUR 2013), volume 8052 of *Lecture Notes in Computer Science*, pages 455–469. Springer, 2013.

12 Claudia Carapelle, Alexander Kartzow, and Markus Lohrey. Satisfiability of ECTL* with Constraints, 2015. submitted for publication, `http://www.eti.uni-siegen.de/ti/veroeffentlichungen/ectl-with-constraints.pdf`.

13 Karlis Cerans. Deciding properties of integral relational automata. In *Proceedings of the 21st International Colloquium on Automata, Languages and Programming (ICALP 1994)*, volume 820 of *Lecture Notes in Computer Science*, pages 35–46. Springer-Verlag, 1994.

14 Stéphane Demri and Morgan Deters. Temporal logics on strings with prefix relation. *Journal of Logic and Computation*, 2015. to appear.

15 Stéphane Demri and Deepak D'Souza. An automata-theoretic approach to constraint LTL. *Information and Computation*, 205(3):380–415, 2007.

16 Stéphane Demri and Régis Gascon. Verification of qualitative Z constraints. *Theoretical Computer Science*, 409(1):24–40, 2008.

17 Stéphane Demri and Ranko Lazić. LTL with the freeze quantifier and register automata. *ACM Transactions on Computational Logic*, 10(3):16:1–16:30, 2009.

18 E. Allen Emerson and Charanjit S. Jutla. The complexity of tree automata and logics of programs. *SIAM Journal on Computing*, 29(1):132–158, 1999.

19 Shigunag Feng, Markus Lohrey, and Karin Quaas. Path-checking for MTL and TPTL over data words. In *Proceedings of the 19th International Conference on Developments in Language Theory (DL 2015)*, 2015. to appear.

20 Régis Gascon. An automata-based approach for CTL* with constraints. *Electronic Notes in Theoretical Computer Science*, 239:193–211, 2009.

21 Alexander Kartzow and Thomas Weidner. Model checking constraint LTL over trees. Technical report, arxiv.org, 2015. `http://arxiv.org/abs/1504.06105`.

22 Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.

23 Carsten Lutz. Description logics with concrete domains – a survey. In *Advances in Modal Logic 4*, pages 265–296. King's College Publications, 2003.

24 Carsten Lutz. Combining interval-based temporal reasoning with general TBoxes. *Artificial Intelligence*, 152(2):235–274, 2004.

25 Carsten Lutz. NEXP TIME-complete description logics with concrete domains. *ACM Transactions on Computational Logic*, 5(4):669–705, 2004.

26 Carsten Lutz and Maja Milicic. A tableau algorithm for description logics with concrete domains and general TBoxes. *Journal of Automated Reasoning*, 38(1-3):227–259, 2007.

27 Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS 1977)*, pages 46–57. IEEE Computer Society, 1977.

28 A. Prasad Sistla and Edmund M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3):733–749, 1985.

29 Wolfgang Thomas. Computation tree logic and regular omega-languages. In *REX Workshop*, pages 690–713, 1988.

30 Moshe Y. Vardi and Larry J. Stockmeyer. Improved upper and lower bounds for modal logics of programs: Preliminary report. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, STOC 1985*, pages 240–251. ACM, 1985.

31 Moshe Y. Vardi and Pierre Wolper. Yet another process logic (preliminary version). In *Logic of Programs*, pages 501–512, 1983.

32 Frank Wolter and Michael Zakharyaschev. Spatio-temporal representation and reasoning based on RCC-8. In *Proceedings of the 7th Conference on Principles of Knowledge Representation and Reasoning (KR2000)*, pages 3–14. Morgan Kaufmann, 2000.

Thinking Algorithmically About Impossibility*

R. Ryan Williams

Computer Science Department, Stanford University
Stanford, CA, USA
rrw@cs.stanford.edu

Abstract

Complexity lower bounds like $\mathsf{P} \neq \mathsf{NP}$ assert impossibility results for all possible programs of some restricted form. As there are presently enormous gaps in our lower bound knowledge, a central question on the minds of today's complexity theorists is *how will we find better ways to reason about all efficient programs?*

I argue that some progress can be made by (very deliberately) thinking *algorithmically* about lower bounds. Slightly more precisely, to prove a lower bound against some class $\mathcal{C}$ of programs, we can start by treating $\mathcal{C}$ as a set of inputs to another (larger) process, which is intended to perform some basic analysis of programs in $\mathcal{C}$. By carefully studying the algorithmic "meta-analysis" of programs in $\mathcal{C}$, we can learn more about the *limitations* of the programs being analyzed.

This essay is mostly self-contained; scant knowledge is assumed of the reader.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes

Keywords and phrases satisfiability, derandomization, circuit complexity

Digital Object Identifier 10.4230/LIPIcs.CSL.2015.14

Category Invited Talk

1 Introduction

We use the term *lower bound* to denote an assertion about the computational intractability of a problem. For example, the assertion "factoring integers of 2048 bits cannot be done with a Boolean circuit of 10^6 gates" is a lower bound which we hope is true (or at least, if the lower bound is false, we hope that parties with sinister motivations have not managed to find this magical circuit).

The general problem of mathematically proving computational lower bounds is a mystery. The stability of modern commerce relies on certain lower bounds being true (most prominently in cryptography and computer security). Yet for practically all of the prominent lower bound problems, we do not know how to begin proving them true – we do not even know *step zero*. (For some major open problems, such as the Permanent versus Determinant problem in arithmetic complexity [15], we do have good candidates for step zero, and possibly step one.) Many present lower bound conjectures may simply be false. In spite of our considerable intuitions about lower bounds, we must admit that our formal understanding of them is awfully weak. This translates to a lack of understanding about algorithms as well.

* Supported in part by NSF CCF-1212372 and a Sloan Fellowship. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

24th EACSL Annual Conference on Computer Science Logic (CSL 2015).
Editor: Stephan Kreutzer; pp. 14–23

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1.1 Barriers

Why are lower bounds so difficult to prove? There are formal reasons, which are often called "complexity barriers." These are theorems which demonstrate that the usual tools for reasoning about computability and lower bounds – such as universal simulators – are simply too abstract to distinguish modes of computation like P and NP. There are three major classes of barriers known.

Relativization, Algebrization, Natural Proofs. Many ways of reasoning about algorithm complexity are equally valid when one adds "oracles" to the computational model: that is, one adds an instruction that can call an *arbitrary* function $O : \{0,1\}^\star \to \{0,1\}$ in one step, as a black box. When a proof of a theorem is true no matter which O is added to the instruction set, we say that the proof "relativizes." A relativizing proof is generally a quite powerful and broadly applicable object. However, relativizing proofs are of limited use in lower bounds: for instance, $\mathsf{P} = \mathsf{NP}$ when some oracles O are added to polynomial-time and nondeterministic-polynomial-time algorithms, but $\mathsf{P} \neq \mathsf{NP}$ when some other oracles O' are added (as proved by Baker, Gill, Solovay [3]). Practically all other open lower bound conjectures exhibit a similar resistance to arbitrary oracles, and a surprisingly large fraction of theorems in complexity theory do relativize.

The more recent "algebrization" barrier [1] teaches a similar lesson, applied to a broader set of algebraic techniques that was designed to get around relativization. (Instead of looking at oracles, they look at a more general algebraic object.) To scale relativization and algebrization, it is necessary to crack open the guts of programs, and reason more closely about their behavior relative to their simple instructions.[1]

The Razborov-Rudich "natural proofs" barrier [19] has a more subtle pedagogical point compared to the other two: informally, they show that strong lower bound proofs *cannot* produce a polynomial-time algorithm for determining whether a given function is hard or easy to compute – otherwise, such an algorithm would (in a formal sense) refute stronger lower bounds that we also believe to hold. It turns out that many lower bound proofs from the 1980s and 1990s have such an algorithm embedded in them.

1.2 Intuition and Counter-Intuition

There are also strong intuitive reasons for why lower bounds are hard to prove. The most common one is that it seems extraordinarily difficult to reason effectively about the infinite set of all possible efficient programs, including programs that we will never see or execute, and argue that none of them can solve an NP-complete problem. Based on this train of thought, some famous computer scientists such as our colleague Donald Knuth have dubbed problems like P versus NP to be "unknowable" [11].

But how difficult is it, really, to reason about all possible efficient programs? Let us give some counter-intuition, which will build up to the point of this article, starting with the observation that while reasoning about lower bounds appears to be difficult, reasoning about *worst-case algorithms* does not suffer from the same appearance. Reasoning computationally about an infinite number of finite objects is commonplace in the analysis of worst-case algorithms. There, we have a function $f : \Sigma^\star \to \Sigma^\star$ in mind, and we prove that some known efficient procedure P outputs $f(x)$ on *all possible* finite inputs x. That is, often in

[1] There are definitely "non-relativizing" and even "non-algebrizing" techniques in complexity theory, but they are a minority; see Section 3.4 in Arora and Barak [2] for more discussion.

algorithm analysis we manage to reason about all possible x, even those x's we will never see or encounter in the real world. The idea is that, if we consider computational problems which

(a) treat their inputs as *programs*,
(b) determine interesting properties of the function computed by the input program, and
(c) have interesting algorithms,

then we can hope to import ideas from the design and analysis of algorithms into the theory of complexity lower bounds. (Yes, this is vague, but it is counter-*intuition*, after all.)

Sanity Check: Computability Theory. We must be careful with this counter-intuition. Which computational problems actually satisfy those three conditions? Undergraduate computability theory (namely, Rice's theorem [20]) tells us that, if x encodes a program that takes arbitrarily long inputs, it is undecidable to determine non-trivial properties of the problem solved by x. The source of this undecidability is the "arbitrary length" of inputs; if x encodes a program that takes only *finitely* many inputs, say only inputs of a fixed length, then we can produce the entire function computed by x, and decide some non-trivial properties of that function. To simplify the discussion (and without significantly losing generality), we might as well think of x as encoding a Boolean circuit over AND, OR, and NOT gates, taking some n bits of input and outputting some m bits. Now, x simply encodes a directed acyclic graph with additional labels on its nodes, and a procedure P operating on x's can be said to be reasoning about the finitary behavior of finite computational objects.

However, one of the major lessons of the theory of NP-hardness is that, while reasoning about arbitrary programs may be undecidable, reasoning about arbitrary circuits *is* often decidable but is still likely to be intractable. Probably the simplest possible circuit analysis problem is: *given a Boolean circuit C, does C compute the all-zeroes function?* This problem is already very difficult to solve; it is equivalent to the NP-complete problem Circuit Satisfiability (which asks if there is some input on which C outputs 1). From this point of view, the assertion $\mathsf{P} \neq \mathsf{NP}$ tells us that arbitrary programs are hard to analyze even over finitely many inputs: we cannot feasibly determine if a given circuit is *trivial* or not. As circuit complexity is inherently tied to P versus NP, the assertion $\mathsf{P} \neq \mathsf{NP}$ appears to have negative consequences for its own provability; this looks depressing. (This particular intuition has been proposed many times before; for instance, the Razborov-Rudich work on "natural proofs" [19] may be viewed as one way to formalize it.)

Slightly Faster SAT Algorithms? The hypothesis $\mathsf{P} \neq \mathsf{NP}$ only says that *very* efficient circuit analysis is impossible. More precisely, for circuits C with k bits of input encoded in n bits, $\mathsf{P} \neq \mathsf{NP}$ means there is no $k^{O(1)} \cdot n^{O(1)}$ time algorithm for detecting if C is satisfiable. There is a giant gap between this sort of bound and the $2^k \cdot n^{O(1)}$ time bound obtained by simple exhaustive search over all possible inputs to the circuit. What if we simply asked for a *non-trivial running time* for detecting the satisfiability of C, something merely faster than exhaustive search?

I would like to argue that finding any asymptotic improvement over 2^k time for Circuit Satisfiability is already a very interesting problem. This is not an obvious point to argue. First off, without any further knowledge of its inner workings, a 1.9^k time algorithm would not be terribly more useful in *practice* than a 2^k one: only for small values of k would one see a difference, and the rest of the instances would remain intractable.[2] Work on worst-case

[2] This attitude is not shared in cryptography, where any improvement in exhaustive search over all keys may be considered a "break" in the cryptosystem.

algorithms for SAT for many years (such as [13, 6, 14, 12, 18, 9, 22, 17, 23, 5], see the survey [7]) was primarily motivated by the intrinsic interest in understanding whether trivial exhaustive search is optimal.

The Non-Black-Box-ness of Circuit SAT Algorithms. There is also a deeper reason to pursue minor improvements in SAT algorithms. Any algorithm for Circuit SAT running in (say) time $1.9^k \cdot n^{O(1)}$ must necessarily provide a "non-relativizing" analysis of the given circuit C, an analysis which relies on the structure and encoding of C. If you were asked to design a Circuit SAT algorithm which could only access C as an *oracle*, obtaining outputs from inputs and no other information, then your algorithm would necessarily require at least 2^k steps in the worst case. The reason is simple: if you are completely blind to the insides of the circuit C, then even a small ($k^{O(1)}$ size) circuit could hide a satisfying k-bit input from you. To thwart you, I may choose a small circuit which only outputs 1 on the "last" input you will call it on, and since you only see 0-outputs, you will need to call the circuit on all 2^k inputs to determine satisfiability. Therefore, a $1.9^k \cdot n^{O(1)}$ time algorithm for Circuit SAT must necessarily use the fully-given representation of the circuit in some critical ways, to work faster than exhaustive search. Even an algorithm running in $O(2^k/k)$ time on circuits of size $O(k)$ would be interesting, for the same reason.[3]

A Possible Road to Circuit Complexity. The ability to analyze a given circuit more efficiently than analyzing a black box suggests a further implication: a Circuit SAT algorithm running faster than exhaustive search could potentially be used to prove a circuit complexity *limitation*. At the very least, if the Circuit SAT problem can be solved faster than exhaustive search on a given collection of circuits $\mathcal{C}$ (some of which encode the all-zero function, and some which do not), then the collection $\mathcal{C}$ *fails* to obfuscate the all-zeroes function from some algorithm running in less than 2^k steps. That is, the assumed Circuit SAT algorithm can "efficiently" distinguish all circuits encoding the all-zeroes function from those circuits which do not; these circuit cannot hide satisfying inputs as well as oracles can. This points to a potential deficiency in $\mathcal{C}$ that the Circuit SAT algorithm takes advantage of. Surprisingly, this intuitive viewpoint can be made formal.

Outline. In the remainder of this article, we first describe some known connections between circuit satisfiability algorithms and circuit complexity lower bounds (Section 2). Then, we turn to a more recent example of how algorithms and lower bounds are tied to each other, in a way that we believe should be of interest to the union of logicians and computer scientists (Section 3). In particular, we reconsider the basic problem of testing circuit functionality via input-output examples, define the *data complexity* as a way of measuring the difficulty of testing, and describe how circuit complexity lower bounds are *equivalent* to data complexity upper bounds. We conclude the article with some hopeful thoughts.

[3] Perhaps you do not believe that Circuit SAT can be solved any faster than $2^k \cdot n^{O(1)}$ steps. This belief turns out to be inessential for the main intuition and the formal theorems that follow. For example, you may instead believe that we can non-deterministically approximate the fraction of satisfying assignments to a k-input circuit of size n in $1.9^k \cdot n^{O(1)}$ time; this is also something that oracle access to a circuit cannot accomplish. Furthermore, if you do not believe even this, then your lack of faith in algorithms requires you to have strong beliefs in the power of Boolean circuits – they would be powerful enough to solve nondeterministic exponential-time problems. See [10, 26].

2 Circuit SAT versus Circuit Complexity

Let us briefly review some relevant notions from the theory of circuit complexity; for more, see the textbook of Arora and Barak ([2], Chapter 6).

Circuit Complexity. A Boolean circuit with n inputs and one output is a directed acyclic graph with n sources and one sink, and labels of AND/OR/NOT on all other nodes. Each circuit computes some finite function $f : \{0,1\}^n \to \{0,1\}$. To compute infinite languages, of the form $L : \{0,1\}^\star \to \{0,1\}$, we extend our computational model to have an infinite family of circuits $\mathcal{F} = \{C_n\}_{n=1}^{\infty}$, where C_n has n inputs and one output. For such a family $\mathcal{F}$, we say that $\mathcal{F}$ *computes* L if for all $x \in \{0,1\}^\star$ we have $C_{|x|}(x) = L(x)$. For a function $s : \mathbb{N} \to \mathbb{N}$, a family $\mathcal{F}$ has *size* $s(n)$ if for all n, the number of nodes (i.e., gates) in C_n is at most $s(n)$. A language L *has polynomial-size circuits* if there is a polynomial $p(n)$ and a family $\mathcal{C}$ of size $p(n)$ that computes L. The class of all languages having polynomial-size circuits is denoted by P/poly. One should think of P/poly as the class of computations for which the minimum "sizes" of computations do not grow considerably with the input length to those computations.

The class P/poly is poorly understood animal. It could be enormously powerful, or it could be fairly weak. It is easy to see that every language over the single alphabet symbol 0 is in P/poly; however, a simple counting argument shows there are undecidable subsets over a single alphabet symbol. Therefore P/poly can "compute" some undecidable languages. In that sense, P/poly is powerful, but this really stems from the fact that the computational model defining P/poly can have infinite-length descriptions. (This observation also shows that traditional thought in computability theory is not going to be very helpful in understanding the power of P/poly.) However, P/poly also looks obviously limited, in another sense: for each input length n, only polynomial-in-n resources need to be spent in order to decide all 2^n inputs of that length. A counting argument shows that for every n, some function $f : \{0,1\}^n \to \{0,1\}$ requires circuits of size exponential-in-n; in fact, *most* functions do.

A prominent question in complexity theory is: *How does* P/*poly relate to the Turing-based classes of classical complexity theory, like* P*,* NP*,* PSPACE*, etc.?* It is pretty easy to see that $\mathsf{P} \subset \mathsf{P/poly}$: every "finite segment" of a polynomial-time algorithm can be simulated with a polynomial-size circuit. It is conjectured that $\mathsf{NP} \not\subset \mathsf{P/poly}$ (which would in turn imply $\mathsf{P} \neq \mathsf{NP}$). But it is an open problem to prove that $\mathsf{NEXP} \not\subset \mathsf{P/poly}$! That is, every language in the *exponential-time version of* NP could in fact have polynomial-size circuits. It looks amazing that a problem like this is still open. Fifty years ago, Hartmanis and Stearns [8] showed that some $O(n^3)$-time computations are more powerful than all $O(n)$-time ones; how is it possible that we can't distinguish exponential time from polynomial size? This open problem demonstrates how truly difficult it is to prove lower bounds on circuit complexity; maybe the infinite circuit model is powerful!

2.1 Enter Circuit SAT

Let's return to thinking about the role of Circuit Satisfiability. We were arguing that a faster algorithm for satisfiability points to a deficiency in the power of circuits, being unable to hide satisfying inputs from algorithms running in less than 2^k steps. We want to say:

The *existence* of a "faster" algorithm A solving Circuit SAT,
for *all* circuits C from a class of circuits $\mathcal{C}$
$\Longrightarrow$
The *existence* of a function $f : \{0,1\}^\star \to \{0,1\}$,
that is not computable by *all* circuit families from that class $\mathcal{C}$

Written this way, the logical quantifiers match up nicely, and the whole idea of using an algorithm to prove a circuit complexity lower bound looks less counter-intuitive.

Indeed, we can say a formal statement as described in the above box. Here is one version:

▶ **Theorem 2.1** ([26, 28]). *Suppose for all polynomials p, satisfiability of circuits with n inputs and $p(n)$ size is decidable in $O(2^n/n^{10})$ time. Then* NEXP $\not\subset$ P/*poly. That is, there are functions computable in nondeterministic exponential time that do not have polynomial-size circuits.*

(The polynomial n^{10} is almost certainly not optimal, but it suffices.) Notice the required improvement over exhaustive search: it would normally take $2^n \cdot p(n)$ time; in order to solve satisfiability fast enough, we need to "divide by an arbitrary polynomial" in the running time. This is a much weaker requirement than bounds like 1.9^n time, which had been the primary focus of researchers.

How Does The Proof Work? In this article, we can only briefly describe the proof of Theorem 2.1; for more technical details, see the surveys [24, 21, 16]. The informal statement in the box above says that a Circuit SAT algorithm A implies a hard function f. One might think that the algorithm A solving Circuit SAT may appear somewhere in the definition of f. That would be a very interesting property, but the proofs that we know do not do this explicitly. Instead, the proofs of Theorem 2.1 proceed by contradiction. We assume:

1. there is a Circuit SAT algorithm A running in $2^n/n^{10}$ time, and
2. every function $f \in$ NEXP has polynomial-size circuits. (It is equivalent to assume that a single function, complete for NEXP under polynomial-time reductions, is computable with polynomial-size circuits.)

These two assumptions are inherently algorithmic in nature: item (1) asserts that Circuit SAT can be solved faster, and item (2) asserts that a huge class of decidable problems can be computed with polynomial-size circuit families. Then we utilize these two assumptions to construct another algorithm which is provably *impossible*. Namely, these assumptions imply every function computable in nondeterministic time 2^n is computable in nondeterministic time $2^n/n^2$, which contradicts the time hierarchy theorem for nondeterminism [29]. At a very high level, the $2^n/n^2$ time algorithm works by:

- nondeterministically *guessing* small circuits for time 2^n computations, asserted to exist by item (2), and
- deterministically *verifying* the correctness of those circuits, using the Circuit SAT algorithm asserted by item (1).

The verification step is a subtle process. If a circuit happens to agree with our nondeterministic 2^n time function, it is not at all clear how we might use a circuit satisfiability call to check that circuit. Roughly speaking, we show that one can guess a small circuit C that is intended to succinctly encode an *accepting computation history* of the nondeterministic 2^n

time computation, and set up a larger circuit D (with C embedded in it) which is unsatisfiable if and only if C does encode an accepting history. This circuit D is carefully constructed so that its number of inputs is essentially n, logarithmic in the running time of the computation it is verifying. Then, a faster circuit satisfiability algorithm can check a 2^n time computation in less than 2^n steps.

While it yields the desired outcome, this style of proof is indirect and feels lacking. It is an interesting open problem to find simpler and/or more informative proofs of this algorithms-to-lower-bounds connection.

Applying the Connections. The framework behind Theorem 2.1 has been generalized so that circuit satisfiability algorithms for various circuit classes $\mathcal{C}$ imply lower bounds for computing functions in NEXP with circuits from $\mathcal{C}$. So far, through the design of new circuit satisfiability algorithms, this framework has led to three unconditional circuit lower bound results:

- NEXP does not have so-called ACC^0 circuits of polynomial size [28],
- $\mathsf{NE}/1 \cap \mathsf{coNE}/1$ (a potentially weaker class) does not have ACC^0 circuits of polynomial size [25], and
- NEXP does not have ACC^0 circuits of polynomial size, augmented with a layer of neurons (linear threshold gates) that connect directly to the inputs [27].

The first and third results were obtained by designing explicit circuit satisfiability algorithms for relevant circuit classes; the second was obtained by sharpening the complexity-theoretic arguments. It is possible that we might prove $\mathsf{NEXP} \not\subset \mathsf{P}/\mathrm{poly}$ without providing a new Circuit SAT algorithm: it might be that the *assumption* $\mathsf{NEXP} \subset \mathsf{P}/\mathrm{poly}$ could imply the existence of algorithms sufficient for proving $\mathsf{NEXP} \not\subset \mathsf{P}/\mathrm{poly}$. (It is known that $\mathsf{NEXP} \subset \mathsf{P}/\mathrm{poly}$ implies faster algorithms for solving some NP problems, but Circuit SAT is not known to be among them.)

3 Circuit Complexity and Testing Circuits With Data

We now turn to a problem related to program verification, and describe an emerging connection to circuit complexity. In practice, programs are often verified by the quick-and-dirty method of trial and error: the program is executed on a suite of carefully chosen inputs, and one checks that the outputs of the program are what is expected. For a given function to compute, it is natural to ask when trial and error can be efficient: when does it suffice to use a small number of input-output examples, and determine correctness of the program?

If we had no constraints on what the program could be, then there would not be much to say about this problem: without any further information, the program is a black box and one would simply have to try all possible inputs to know for sure. But in testing, we're never given a program as a black box; we know *something* about it, such as its size. Could side information such as program size be useful for the testing problem?

Recently with Brynmor Chapman [4], we have proposed the general problem of *data design*. Suppose we are given a function $f : \{0,1\}^\star \to \{0,1\}$, and a class $\mathcal{C}$ of size-s circuits. The task of data design is to select a small suite of input-output *test data* that can be used to determine whether a given n-input circuit C from $\mathcal{C}$ computes f restricted to n-bit inputs. More formally, the *data complexity* of f (as a function of the circuit size s) is defined to be the minimum number of input-output examples such that, for any n-input circuit C of size s, one can determine with certainty whether C computes f (on all n-bit inputs), by calling C on the examples. Every function f that depends on all of its inputs has data complexity

$O(2^s)$: the test suite may contain all possible input-output pairs for f on all input lengths $n = 1, \ldots, s$. When can test suites be made small?

While the data design problem certainly has practical motivation, we are interested in the problem due to its intriguing inversion of the roles of program and input. The circuit computing f is the *input* to the data design problem; the *program* for testing the circuit is the collection of input-output examples. We have uncovered a surprising correspondence between upper bounds on data design and lower bounds on circuit complexity. Generally speaking, designing good suites of data for testing whether $\mathcal{C}$-circuits compute f is *equivalent* to proving $\mathcal{C}$-circuit lower bounds on computing f. For example:

▶ **Corollary 3.1** ([4]). *A function f is in $\mathsf{P}/poly$ if and only if for some $\varepsilon > 0$, the data complexity of testing circuits for f is greater than 2^{s^ε} for almost every s.*

So if we wanted to prove that (for example) that $\mathsf{NEXP} \not\subset \mathsf{P}/\mathrm{poly}$, it would be necessary and sufficient to design test suites of subexponential data complexity for a function in NEXP.

Intuitively, such a correspondence is possible because the circuit design problem and the data design problems work with similar types of unknowns. The circuit designed must compute f on all n-bit inputs, and the data designed must test the functionality of f for all s-size circuits. There are two parts to the equivalence:

- If f has an n-input circuit of size at most s, then standard arguments show that our test suite essentially needs all 2^n input-output pairs for f on n bits, to distinguish "good" circuits computing f from slightly different circuits which give the wrong answer on one input.
- If f does not have an n-input circuit of size s, then arguments from the theory of zero-sum games show that the test suite only needs $\mathrm{poly}(s)$ examples on n-inputs in order to test all circuits of size at most s/n.

Putting the two items together, one can show that upper bounds on data design for f are equivalent to lower bounds on the circuit complexity of f. Therefore, reliable exhaustive circuit testing and proving circuit lower bounds are deeply related tasks, in ways that are not fully understood yet. Not only would small test suites detect errors efficiently, they would also be useful for formal verification. Assuming the circuit being tested is in the appropriate class $\mathcal{C}$, passing a test suite would be a proof of correctness on all inputs. In turn, proving that a small test suite works is equivalent to proving a limitation on $\mathcal{C}$. This "constructive" algorithmic viewpoint on lower bounds is still in its early stages of development, and it remains to be seen how effectively one can prove new (or old!) lower bounds with it.

4 Conclusion

I believe that knowledge from all areas of theoretical computer science could contribute significantly to the general projects outlined here. Computer scientists will have to develop new methods of argument in order to make a serious dent in the major lower bound problems, and it is worth trying every sort of reasonable argument we can think of (at least once). Perhaps the logic side of computer science will provide some of these new proof methods.

References

1 Scott Aaronson and Avi Wigderson. Algebrization: A new barrier in complexity theory. *ACM TOCT*, 1, 2009.

2 Sanjeev Arora and Boaz Barak. *Computational Complexity – A Modern Approach*. Cambridge University Press, 2009.

3 Theodore Baker, John Gill, and Robert Solovay. Relativizations of the P =? NP question. *SIAM J. Comput.*, 4(4):431–442, 1975.

4 Brynmor Chapman and Ryan Williams. The circuit-input game, natural proofs, and testing circuits with data. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science, ITCS 2015, Rehovot, Israel, January 11-13, 2015*, pages 263–270, 2015.

5 Ruiwen Chen, Valentine Kabanets, Antonina Kolokolova, Ronen Shaltiel, and David Zuckerman. Mining circuit lower bound proofs for meta-algorithms. *Computational Complexity*, 24(2):333–392, 2015. See CCC'14.

6 Evgeny Dantsin. Two propositional proof systems based on the splitting method. *Zapiski Nauchnykh Seminarov LOMI*, 105:24–44, 1981.

7 Evgeny Dantsin and Edward A. Hirsch. Worst-case upper bounds. In *Handbook of Satisfiability*, pages 403–424. 2009.

8 Juris Hartmanis and Richard Stearns. On the computational complexity of algorithms. *Trans. Amer. Math. Soc. (AMS)*, 117:285–306, 1965.

9 Edward A. Hirsch. New worst-case upper bounds for SAT. *J. Autom. Reasoning*, 24(4):397–420, 2000.

10 Russell Impagliazzo, Valentine Kabanets, and Avi Wigderson. In search of an easy witness: Exponential time vs. probabilistic polynomial time. *J. Comput. Syst. Sci.*, 65(4):672–694, 2002.

11 Donald E. Knuth. Personal communication, 2015.

12 Oliver Kullmann and Horst Luckhardt. Deciding propositional tautologies: Algorithms and their complexity. Technical report, Fachbereich Mathematik, Johann Wolfgang Goethe Universität, 1997.

13 Burkhard Monien and Ewald Speckenmeyer. 3-satisfiability is testable in $O(1.62^r)$ steps. Technical Report Bericht Nr. 3/1979, Reihe Theoretische Informatik, Universität-Gesamthochschule-Paderborn, 1979.

14 Burkhard Monien and Ewald Speckenmeyer. Solving satisfiability in less than 2^n steps. *Discrete Applied Mathematics*, 10(3):287–295, 1985.

15 Ketan Mulmuley. The GCT program toward the *P* vs. *NP* problem. *Commun. ACM*, 55(6):98–107, 2012.

16 Igor Oliveira. Algorithms versus circuit lower bounds. Technical Report TR13-117, ECCC, September 2013.

17 Ramamohan Paturi, Pavel Pudlák, Michael E. Saks, and Francis Zane. An improved exponential-time algorithm for k-SAT. *JACM*, 52(3):337–364, 2005. (See also FOCS'98).

18 Pavel Pudlák. Satisfiability – algorithms and logic. In *Mathematical Foundations of Computer Science 1998, 23rd International Symposium, (MFCS'98)*, pages 129–141, 1998.

19 Alexander Razborov and Steven Rudich. Natural proofs. *J. Comput. Syst. Sci.*, 55(1):24–35, 1997.

20 H. G. Rice. Classes of recursively enumerable sets and their decision problems. *Trans. Amer. Math. Soc.*

21 Rahul Santhanam. Ironic complicity: Satisfiability algorithms and circuit lower bounds. *Bulletin of the EATCS*, 106:31–52, 2012.

22 Uwe Schöning. A probabilistic algorithm for k-SAT based on limited local search and restart. *Algorithmica*, 32(4):615–623, 2002.

23 Magnus Wahlström. *Algorithms, Measures, and Upper Bounds for Satisfiability and Related Problems*. PhD thesis, Linköping University, 2007.

24 Ryan Williams. Guest column: a casual tour around a circuit complexity bound. *ACM SIGACT News*, 42(3):54–76, 2011.

25 Ryan Williams. Natural proofs versus derandomization. In *STOC*, pages 21–30, 2013.

26 Ryan Williams. Improving exhaustive search implies superpolynomial lower bounds. *SIAM J. Comput.*, 42(3):1218–1244, 2013. See also STOC'10.

27 Ryan Williams. New algorithms and lower bounds for circuits with linear threshold gates. In *STOC*, pages 194–202, 2014.

28 Ryan Williams. Nonuniform ACC circuit lower bounds. *JACM*, 61(1):2, 2014. See also CCC'11.

29 Stanislav Žák. A Turing machine time hierarchy. *Theoretical Computer Science*, 26(3):327–333, 1983.

Simple Parsimonious Types and Logarithmic Space

Damiano Mazza

CNRS, LIPN UMR 7030 Université Paris 13, Sorbonne Paris Cité
F-93430, Villetaneuse, France
Damiano.Mazza@lipn.univ-paris13.fr

Abstract

We present a functional characterization of deterministic logspace-computable predicates based on a variant (although not a subsystem) of propositional linear logic, which we call parsimonious logic. The resulting calculus is simply-typed and contains no primitive besides those provided by the underlying logical system, which makes it one of the simplest higher-order languages capturing logspace currently known. Completeness of the calculus uses the descriptive complexity characterization of logspace (we encode first-order logic with deterministic closure), whereas soundness is established by executing terms on a token machine (using the geometry of interaction).

1998 ACM Subject Classification F.4.1 Mathematical Logic, F.1.3 Complexity Measures and Classes

Keywords and phrases implicit computational complexity, linear logic, geometry of interaction

Digital Object Identifier 10.4230/LIPIcs.CSL.2015.24

1 Introduction

Implicit computational complexity (ICC) is a research field at the intersection of logic, computational complexity and the theory of programming languages, arising from the seminal contributions of, among others, Bellantoni and Cook [3], Leivant and Marion [18], and Jones [16]. ICC may be thought of as the proof-theoretic counterpart of descriptive complexity [15], which is based on model theory instead. They both invoke logic as a guideline for understanding the nature of complexity classes, seeking alternatives to the notion of complete problem which is proper of structural complexity theory.

Linear logic has proved to be quite valuable for ICC, spurring a fruitful line of research [12, 14, 25, 9] which we continue with the present paper: we show how an affine propositional logical system characterizes in a natural way the class L of logspace computable predicates. Such a logical system stems from previous work by the author on the infinitary affine λ-calculus [19, 20]. In particular, the latter work introduced the so-called *parsimonious stratified λ-calculus*, which was shown to capture (non-uniform) polytime computation. In that paper, parsimony was considered merely as a restriction to be added on top of stratification in order to keep the complexity under control. Later, the author realized that parsimony has an independent logical meaning, *i.e.*, it corresponds to a well-defined logical system which is a variant (but not a subsystem) of linear/affine logic.

In its intuitionistic form, parsimonious logic is multiplicative affine logic (*i.e.*, with linear implication $\multimap$, multiplicative conjunction $\otimes$ and free weakening; categorically, the free SMCC with terminal unit) endowed with an exponential modality satisfying what we call *Milner's law* $!A \cong A \otimes !A$. The implication $!A \multimap A \otimes !A$, sometimes called *absorption*, holds in linear logic but its converse, which we deem *co-absorption*, does not.[1] It does hold in

[1] To be fair, in linear logic one should look for $!A \cong (A \,\&\, 1) \otimes !A$. Nevertheless, although implications in

24th EACSL Annual Conference on Computer Science Logic (CSL 2015).
Editor: Stephan Kreutzer; pp. 24–40
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

differential linear logic (it is the type of the derivative operator), but it is not the inverse of absorption. Therefore, Milner's law is not verified in any known variant of linear/affine logic.

From the computational point of view, Milner's law asserts that $!A$ is the type of streams of type A: absorption is "pop" and co-absorption is "push". The fact that these operations are inverses of each other is why we speak of (infinite) streams rather than (finite) stacks. Accordingly, the λ-calculus arising from parsimonious logic natively supports streams. Remember that parsimony arises from an infinitary affine calculus, so we are in principle capable of dealing with truly infinite streams. This yields *non-uniform* computation (in the same sense as circuit families) and is the object of [20, 21]. However, in the present paper we focus on *uniform* computation, which means that the streams will be morally finite: we will consider only terms of the form $u_1 :: \cdots :: u_n :: !t$ where $!t \equiv t :: !t$, *i.e.*, the stream is ultimately constant. In [21], it is proved that the non-uniform simply-typed parsimonious λ-calculus captures $\mathsf{L/poly}$ (non-uniform logspace). Although closely related, this does not imply the results presented here, it rather complements them.

The question we address here is: what is the expressive power of the simply-typed parsimonious λ-calculus? More precisely, if $\mathsf{Str} := !(o \multimap o) \multimap !(o \multimap o) \multimap o \multimap o$ is the affine version of the type of Church binary strings and $\mathsf{Bool} := o \otimes o \multimap o \otimes o$ is the affine type of Booleans, what languages are decidable by simply-typed parsimonious terms of type $\mathsf{Str}[A] \multimap \mathsf{Bool}$?[2] If we call PL the class of such languages, as anticipated above we have:

▶ **Theorem 1.** $\mathsf{PL} = \mathsf{L}$.

By contrast, the analogous question for the usual simply-typed λ-calculus, or for propositional linear logic, lacks to our knowledge such a straightforward, standard answer. This, we believe, supports the claim that parsimony is an interesting and natural notion.

The proof of Theorem 1 is of course in two parts. Completeness ($\mathsf{L} \subseteq \mathsf{PL}$, Sect. 3) is shown by programming in PL the descriptive complexity characterization of L (*i.e.*, first-order logic with deterministic transitive closure). The non-trivial part is, essentially, solving reachability for directed forests, a paradigmatic L-complete problem.

The main ingredient of soundness ($\mathsf{PL} \subseteq \mathsf{L}$, Sect. 5) consists of proof nets. These allow turning the normalization of terms into the execution of an automaton, Danos and Regnier's interaction abstract machine (IAM) [8], based on Girard's geometry of interaction (GoI) [11]. In its IAM formulation, the GoI computes the normal form of a proof net π by considering a token traveling through the nodes of π, instead of applying rewriting rules. To control the movements of the token, the machine has to keep track of a certain amount of information, consisting of two stacks S and B. The stack S is binary, and its length is bounded by the height of the types of π. The length of B is bounded by the *depth* of π (the maximum number of nested exponential modalities in the types of π) but, in linear logic, its elements may be quite complex. In parsimonious logic, the elements of B are just integers: they correspond to positions within streams. Therefore, running the IAM on a proof net π of size s, height h and depth d needs space $O(\log s + h + d \log m)$: $\log s$ is for the token's position, h for the S stack and $d \log m$ for the B stack, if m is the largest integer stored in it during execution.

The interesting case is when π is the translation of $t\,\underline{w}$, with $t : \mathsf{Str}[A] \multimap \mathsf{Bool}$ and $\underline{w}$ a Church string. Then, s is $O(|w|)$, whereas h and d are $O(1)$ (they only depend on A, which is fixed). Therefore, if we manage to prove that m is polynomial in s, we have a logarithmic

both directions are provable, they are not inverses of each other.

[2] $\mathsf{Str}[A]$ denotes Str where the base type o is replaced by an arbitrary simple type A. In the absence of polymorphism, it is common to allow such type expansions.

bound in $|w|$, as desired. Such a bound on m may be proved directly, as done in [21], or by combining the bound we previously gave in [20] with a nice result of Gaboardi, Roversi and Vercelli [10], which is the strategy we adopt here.

Related work

Implicit characterizations of L abound: of recursive-theoretic nature [22, 17], using imperative languages [16, 4] and higher-order languages [24, 25, 6]. Of these, only the latter are immediately comparable to our work. Another paper explicitly relating streams and logarithmic space is [23], which however does not have much of a connection with our work: the authors consider there corecursive definitions, *i.e.*, algorithms on infinite streams (as opposed to finite strings) and the space complexity they refer to is not the usual decision problem complexity.

The GoI plays a key role in both [25, 6]. The difference here is not so much in the use of the GoI, which is quite similar, but in the underlying programming language: in that work, the author(s) take the standpoint that the fundamental primitive of sublinear space computation is interaction (a point of view already taken in [24]) and forge their programming language around this. This leads, for instance, to the use of non-standard types for encoding strings, namely $\mathsf{Nat} \multimap \mathsf{Three}$ (a binary string x is seen as a function mapping i to x_i or to $\bot$ if $i \geq |x|$), whereas the language of [24] has an explicit list type.

With respect to the above, we believe that the highlight of our characterization is that it is closer to the original spirit of applying linear logic to ICC [12]: it is purely logical (there is no primitive datatype) and employs standard types. Our characterization also improves on previous ones in terms of simplicity: the types of [25] include full polymorphism and indexed exponential modalities, whereas the categorical construction of [6], while elegant, also yields a sort of indexed exponential modality in types, making type inference not straightforward (see [7]). By contrast, our calculus is simply-typed, has only 9 typing rules (Fig. 1) which are essentially syntax-directed, so type inference is easier. Programming is of course restricted but, as hopefully showcased by Sect. 3, quite reasonably so if we consider that all programs must run in logarithmic space.

2 The Parsimonious Lambda-Calculus

2.1 Terms and reduction

We let a (resp. x) range over a countably infinite set of *affine* (resp. *exponential*) variables. An *occurrence* of exponential variable is of the form x_i, where $i \in \mathbb{N}$. Occurrences of exponential variables are naturally ordered by $x_i \leq y_j$ whenever $x = y$ and $i \leq j$. Let Θ be a set of occurrences of exponential variables. We write $\uparrow\Theta$ for the upward closure of Θ. We denote by $\Theta \setminus \{x\}$ the set obtained from Θ by removing all occurrences of the form x_i (if any).

Parsimonious terms belong to the grammar

$$t, u ::= a \mid \lambda a.t \mid tu \mid \mathsf{let}\, a \otimes b = u \,\mathsf{in}\, t \mid t \otimes u \mid x_i \mid \mathsf{let}\, !x = u \,\mathsf{in}\, t \mid !t \mid t :: u.$$

However, they obey several constraints on how variables may appear, so we introduce them by means of a well-forming relation $\Theta; \Phi \rhd t$, where t is a (parsimonious) term, Φ is the set of its free affine variables and Θ is the set of its free *virtual* occurrences of exponential variables (which may be infinite):

- $\emptyset; \{a\} \rhd a$ and $\{x_i\}; \emptyset \rhd x_i$ always hold;
- if $\Theta; \Phi \rhd t$, then $\Theta; \Phi \setminus \{a\} \rhd \lambda a.t$;

- if $\Theta;\Phi \rhd t$ and $\Theta';\Phi' \rhd u$ and $\Theta \cap \Theta' = \Phi \cap \Phi' = \emptyset$, then $\Theta \cup \Theta';\Phi \cup \Phi' \rhd tu$, $\Theta \cup \Theta';\Phi \cup \Phi' \rhd t \otimes u$, $\Theta \cup \Theta';\Phi \cup \Phi' \rhd t :: u$, $\Theta \cup \Theta';\Phi \setminus \{a,b\} \cup \Phi' \rhd \mathsf{let}\, a \otimes b = u \,\mathsf{in}\, t$ and $\Theta \setminus \{x\} \cup \Theta';\Phi \cup \Phi' \rhd \mathsf{let}\, !x = u \,\mathsf{in}\, t$;
- if $\Theta;\emptyset \rhd t$ and every exponential variable has at most one occurrence in Θ, then $\uparrow\Theta;\emptyset \rhd !t$.

Let $\Theta;\Phi \rhd t$. We denote by t^{++} the term obtained from t by substituting each free occurrence x_i with x_{i+1}. We have $\Theta^{++};\Phi \rhd t^{++}$, where Θ^{++} is defined in the obvious way.

Terms of the form $!t$ are called *boxes*. Apart from disallowing free affine variables, the main purpose of the well-forming condition for boxes is to ensure that, if x occurs free in a parsimonious term t, then it occurs free in at most one box and at most once therein and, moreover, the index of such an occurrence, denoted by $\mathrm{maxind}_x(t)$, is the greatest in t.

Streams are terms generated by $\mathbf{u} ::= !t \mid t :: \mathbf{u}$. *Structural equivalence* is the contextual closure of the equation $!t \equiv t :: !(t^{++})$. This justifies the name "stream": the term $\mathbf{u} := u_1 :: \cdots :: u_n :: !t$ is morally an infinite stream $u_1 :: \cdots :: u_n :: t :: t^{++} :: (t^{++})^{++} :: \cdots$. Accordingly, given $i \in \mathbb{N}$, we define $\mathbf{u}(i)$ to be u_{i+1} if $i < n$ and $t^{++(i-n \text{ times})}$ if $i \geq n$. We also define $|\mathbf{u}| := n$.

We now define reduction. To avoid the commutative rules induced by the presence of let binders, we use Accattoli's contextual approach (see for instance [1]). We define let-*contexts* as

$$L ::= [\cdot] \mid \mathsf{let}\, \mathsf{p} = t \,\mathsf{in}\, L,$$

where p stands for $a \otimes b$ or $!x$. We denote by $L[t]$ the substitution of the term t for the hole $[\cdot]$ in the let-context L. We may now introduce the reduction rules:

$$\begin{array}{rcl} L[(\lambda a.t)]u & \to_\beta & L[t[u/a]] \\ \mathsf{let}\, a \otimes b = L[u \otimes v] \,\mathsf{in}\, t & \to_\otimes & L[t[u/a, v/b]] \\ \mathsf{let}\, !x = L[\mathbf{u}] \,\mathsf{in}\, t & \to_! & L[t\{\mathbf{u}/x\}] \end{array}$$

Modulo the presence of let-contexts, the rules β and $\otimes$ are standard. In the ! rule, in case x appears in a box in t, we require that $\mathrm{maxind}_x(t) \geq |\mathbf{u}|$. If x does not appear in a box, the rule may always be applied. In any case, $t\{\mathbf{u}/x\}$ stands for t in which every x_i is substituted by $\mathbf{u}(i)$. All rules are readily verified to preserve parsimony.

One-step reduction, denoted by $\to$, is defined as the contextual closure of the above rules, plus closure under structural equivalence, *i.e.*, $t \equiv t' \to u' \equiv u$ implies $t \to u$.

Reduction may be shown to be confluent. However, termination is not guaranteed: if we let $\Delta := \lambda !x.x_0 !x_1$ and $\Omega := \Delta !\Delta$, we have $\Omega \equiv \Delta(\Delta :: !\Delta) \to \Omega$. In fact, the untyped parsimonious λ-calculus is Turing-complete. This follows from observing that, although parsimony seems to exclude general fixpoint combinators, we do have *affine* fixpoints and these are enough to encode partial recursive functions, because the minimization scheme is an affine recurrence.[3]

Note how the syntax allows one to recover unambiguously whether a variable is affine or exponential. For this reason, we will occasionally use any letter to denote any kind of variable. It will also be convenient to use the abbreviations

$$\lambda a \otimes b.t := \lambda c.\mathsf{let}\, a \otimes b = c \,\mathsf{in}\, t \qquad \lambda !x.t := \lambda a.\mathsf{let}\, !x = a \,\mathsf{in}\, t,$$

as well as combinations such as $\lambda a \otimes !x.t := \lambda c.\mathsf{let}\, a \otimes d = c \,\mathsf{in}\, \mathsf{let}\, !x = d \,\mathsf{in}\, t$.

[3] The function $g(\overline{x}) := (\mu y.f(\overline{x}, y) = 0)$ may be defined as $g(\overline{x}) := h(0, \overline{x})$ where $h(n, \overline{x}) := \mathsf{if}\, (f(\overline{x}, n) = 0) \,\mathsf{then}\, n \,\mathsf{else}\, h(n+1, \overline{x})$. This recursive definition is affine because h appears only once on the right.

$$\frac{}{\Gamma;\Delta,a:A\vdash a:A}\ \mathsf{ax} \qquad \frac{\Gamma;\Delta,a:A\vdash t:B}{\Gamma;\Delta\vdash\lambda a.t:A\multimap B}\ \multimap\mathsf{I} \qquad \frac{\Gamma;\Delta\vdash t:A\multimap B\quad\Gamma';\Delta'\vdash u:A}{\Gamma,\Gamma';\Delta,\Delta'\vdash tu:B}\ \multimap\mathsf{E}$$

$$\frac{\Gamma;\Delta\vdash t:A\quad\Gamma';\Delta'\vdash u:B}{\Gamma,\Gamma';\Delta,\Delta'\vdash t\otimes u:A\otimes B}\ \otimes\mathsf{I} \qquad \frac{\Gamma';\Delta'\vdash u:A\otimes B\quad\Gamma;\Delta,a:A,b:B\vdash t:C}{\Gamma,\Gamma';\Delta,\Delta'\vdash\mathsf{let}\,a\otimes b=u\,\mathsf{in}\,t:C}\ \otimes\mathsf{E}$$

$$\frac{;\overline{a}:\Gamma\vdash t:A}{\overline{x}:\Gamma;\vdash !t[\overline{x}_0/\overline{a}]:!A}\ !\mathsf{I} \qquad \frac{\Gamma';\Delta'\vdash u:!A\quad\Gamma,x:A;\Delta\vdash t:C}{\Gamma,\Gamma';\Delta,\Delta'\vdash\mathsf{let}\,!x=u\,\mathsf{in}\,t:C}\ !\mathsf{E}$$

$$\frac{\Gamma,x:A;\Delta,a:A\vdash t:C}{\Gamma,x:A;\Delta\vdash t^{x++}[x_0/a]:C}\ \mathsf{abs} \qquad \frac{\Gamma;\Delta\vdash t:A\quad\Gamma';\Delta'\vdash u:!A}{\Gamma,\Gamma';\Delta,\Delta'\vdash t::u:!A}\ \mathsf{coabs}$$

Figure 1 The simply typed parsimonious calculus **PL**.

2.2 Simple types

The *simple types* are the formulas of intuitionistic propositional linear logic, generated by

$$A,B ::= o \mid A\multimap B \mid A\otimes B \mid !A,$$

where o is a ground type (we consider only one, although of course our results hold for any number of ground types). If A and B are types, we denote by $A[B]$ the type obtained by replacing every occurrence of o in A with B.

The type system we consider, which we call **PL** (for *parsimonious logic*), is defined in Fig. 1. In the abs rule, t^{x++} is defined like t^{++} but only the free occurrences of x are re-indexed. A straightforward induction on the last rule of the derivation gives

▶ **Lemma 2.** *Let* $\Gamma;\Delta\vdash t:A$. *Then:*
parsimony: *t is parsimonious;*
typical ambiguity: *for any type B, $\Gamma[B];\Delta[B]\vdash t:A[B]$.*

In fact, the type assignment is an instance of the Curry-Howard correspondence: if we forget term annotations, the type derivations are proofs in natural deduction and reduction of terms corresponds to normalization. The underlying logical system is the parsimonious logic mentioned in the introduction. The isomorphism $!A\cong A\otimes !A$ is realized by the terms

$$\lambda!x.x_0\otimes !x_1 : !A\multimap A\otimes !A \qquad\qquad \lambda a\otimes !x.a :: !x_0 : A\otimes !A\multimap !A,$$

which use absorption and co-absorption, respectively.

In the sequel, we will sometimes write $A[]$ for $A[B]$ when the type $B\neq o$ is unimportant. This lack of information is harmless for composition: point 2 of Lemma 2 guarantees that terms of type $A[X]\multimap B$ and $B[Y]\multimap C$ may be composed to yield $A[X[Y]]\multimap C$. The only delicate point is iteration (see below), which requires a *flat* typing, *i.e.*, of the form $A\multimap A$.

3 Simply-typed Parsimonious Programming

3.1 Basic data types

The types Nat, Bool and Str of unary (Church) integers, Booleans and binary strings, respectively, are defined in Fig. 2, together with the encoding of integers and Booleans. For binary strings, the encoding is similar: if $w=w_1\cdots w_n\in\mathbb{W}$, we have

$$\underline{w} := \lambda!s^0.\lambda!s^1.\lambda z.s_0^{w_1}(\ldots s_i^{w_n}z\ldots),$$

$\mathsf{Nat} := !(o \multimap o) \multimap o \multimap o$, $\mathsf{Bool} := o \otimes o \multimap o \otimes o$, $\mathsf{Str} := !(o \multimap o) \multimap !(o \multimap o) \multimap o \multimap o$

$\underline{n}$	$:=$	$\lambda!s.\lambda z.s_0(\dots s_{n-1} z \dots)$	$: \mathsf{Nat}$
succ	$:=$	$\lambda n.\lambda!s.\lambda z.s_0(n\,!(s_1)\,z)$	$: \mathsf{Nat} \multimap \mathsf{Nat}$
pred	$:=$	$\lambda n.\lambda!s.\lambda z.n\,((\lambda a.a) :: !s_0)\,z$	$: \mathsf{Nat} \multimap \mathsf{Nat}$
dup	$:=$	$\lambda n.\mathsf{It}(n, \lambda m_1 \otimes m_2.(\mathsf{succ}\ m_1) \otimes (\mathsf{succ}\ m_2), \underline{0} \otimes \underline{0})$	$: \mathsf{Nat}[] \multimap \mathsf{Nat} \otimes \mathsf{Nat}$
store	$:=$	$\lambda n.\mathsf{It}(n, \lambda!x.!(\mathsf{succ}\ x_0), !\underline{0})$	$: \mathsf{Nat}[] \multimap !\mathsf{Nat}$
$\mathsf{tt} := \lambda c \otimes d.c \otimes d$,		$\mathsf{ff} := \lambda c \otimes d.d \otimes c$	$: \mathsf{Bool}$
not	$:=$	$\lambda b.\lambda c \otimes d.b(d \otimes c)$	$: \mathsf{Bool} \multimap \mathsf{Bool}$
xor	$:=$	$\lambda b.\lambda b'.\lambda c.b(b'c)$	$: \mathsf{Bool} \multimap \mathsf{Bool} \multimap \mathsf{Bool}$
and	$:=$	$\lambda b.\lambda b'.\mathsf{let}\ c \otimes d = b(b' \otimes \mathsf{ff})\ \mathsf{in}\ c$	$: \mathsf{Bool}[] \multimap \mathsf{Bool} \multimap \mathsf{Bool}$
len	$:=$	$\lambda w.\mathsf{It}(w, \mathsf{succ}, \mathsf{succ}, \underline{0})$	$: \mathsf{Str}[] \multimap \mathsf{Nat}$
shift	$:=$	$\lambda!x.!x_1$	$: !A \multimap !A$
toStrm	$:=$	$\lambda w.\mathsf{It}(w, \lambda s.(\mathsf{ff} :: s), \lambda s.(\mathsf{tt} :: s), !\mathsf{ff})$	$: \mathsf{Str}[] \multimap !\mathsf{Bool}$
leq	$:=$	$\lambda m.\lambda n.\mathsf{let}\ !x = \mathsf{It}(n, \mathsf{shift}, \mathsf{It}(m, \lambda s.(\mathsf{ff} :: s), !\mathsf{tt}))\ \mathsf{in}\ x_0$	$: \mathsf{Nat}[] \multimap \mathsf{Nat}[] \multimap \mathsf{Bool}$
isOne	$:=$	$\lambda w.\lambda n.\mathsf{let}\ !x = \mathsf{It}(n, \mathsf{shift}, \mathsf{toStrm}\ w)\ \mathsf{in}\ x_0$	$: \mathsf{Str}[] \multimap \mathsf{Nat}[] \multimap \mathsf{Bool}$

Figure 2 Data types, encodings and basic functions.

where the j-th occurrence of s^0 from the left has index $j-1$, and similarly for s^1. For instance, $\underline{001} = \lambda!s^0.\lambda!s^1.\lambda z.s^0_0(s^0_1(s^1_0 z))$.

The type Nat supports iteration $\mathsf{It}(n, \mathsf{step}, \mathsf{base}) := n\,!(\mathsf{step}')\,\mathsf{base}$, typed as:

$$\frac{;\Delta \vdash \mathsf{step} : A \multimap A \quad \Gamma;\Sigma \vdash \mathsf{base} : A}{\Gamma, \Delta';\Sigma, n : \mathsf{Nat}[A] \vdash \mathsf{It}(n, \mathsf{step}', \mathsf{base}) : A}$$

where Δ' and step' are the results of systematically replacing linear variables by exponential ones. Note that the type of step must be flat.

Unary successor and predecessor are implemented as in Fig. 2. Since their types are flat, they may be iterated to obtain addition and subtraction, of type $\mathsf{Nat}[] \multimap \mathsf{Nat} \multimap \mathsf{Nat}$. This is again flat with respect to the second argument, so a further iteration on addition leads to multiplication, of type $\mathsf{Nat}[] \multimap \mathsf{Nat}[] \multimap \mathsf{Nat}$. Unary integers are duplicable and storable as shown in Fig. 2. Using addition, multiplication, subtraction and duplication we may represent any polynomial with integer coefficients as a closed term of type $\mathsf{Nat}[] \multimap \mathsf{Nat}$.

These constructions can all be extended to the type Str, which also supports iteration, flat successors and predecessor, concatenation, and is duplicable and storable.

For the Booleans, we adopt the multiplicative type Bool used in [26]. This type too is duplicable and storable. An advantage of multiplicative Booleans is that they support flat exclusive-or in addition to flat negation (see Fig. 2). On the other hand, conjunction (and disjunction) has one non-flat argument (see again Fig. 2). This would be the case of exclusive-or too if we had chosen the traditional Boolean type $o \multimap o \multimap o$. We write $\mathsf{if}\ b\ \mathsf{then}\ t\ \mathsf{else}\ u$ for $\mathsf{let}\ c \otimes \_ = b\,(t \otimes u)\ \mathsf{in}\ c$, which has type A if $t, u : A$ and $b : \mathsf{Bool}[A]$.

In the sequel we will abusively use affine variables of duplicable types non-linearly, *e.g.*, if $n : \mathsf{Nat}[]$ we write $n \otimes n$ meaning $\mathsf{let}\ n' \otimes n'' = \mathsf{dup}[n]\ \mathsf{in}\ n' \otimes n''$. Similarly, if a term step contains a free affine variable a of storable type A, we will abusively consider the result of its iteration to still have a free variable $a : A[]$ (instead of an exponential variable of type $!A$), by implicitly composing with store.

It will be useful to consider for loops, derived from iteration. Given $\mathsf{step}[i] : A \multimap A$ containing a free affine variable $i : \mathsf{Nat}[]$, we define $\mathsf{step}_+ := \lambda!j \otimes a.!(\mathsf{succ}\ j_1) \otimes (\mathsf{step}[j_0/i]\ a) :$

$$
\begin{aligned}
\mathsf{strmToW} &:= \lambda!x.\lambda m.m\,!(\mathsf{if}\ x_0\ \mathsf{then}\ \mathsf{succ}^1\ \mathsf{else}\ \mathsf{succ}^0)\,\underline{\epsilon} : !\mathsf{Bool}[] \multimap \mathsf{Nat}[] \multimap \mathsf{Str}\\
\mathsf{forall} &:= \lambda w.\mathsf{It}(w, \lambda b.\mathsf{ff}, \lambda b.b, \mathsf{tt}) : \mathsf{Str}[] \multimap \mathsf{Bool}\\
\mathsf{Univ} &:= \lambda!R.\lambda m.\mathsf{forall}(\mathsf{strmToW}(\mathsf{for}\ k < m\ \mathsf{from}\ !\mathsf{ff}\ \mathsf{do}\ \lambda s.(R\,m\,k) :: s))\\
\mathsf{mkDep}_R &:= \mathsf{if}\ (R\,m\,i\,j)\ \mathsf{then}\ \lambda b \otimes !x \otimes !y.(\mathsf{xor}\,b\,x_0) \otimes !x_1 \otimes \mathsf{ff} :: !y_0\\
&\qquad\qquad \mathsf{else}\ \lambda b \otimes !x \otimes !y.b \otimes !x_1 \otimes x_0 :: !y_0\\
\mathsf{rev} &:= \lambda s.\mathsf{let}\ s' \otimes \_ = \mathsf{It}(m, \lambda!x \otimes !y.(y_0 :: !x_0) \otimes !y_1, !\mathsf{ff} \otimes s)\ \mathsf{in}\ s' : !\mathsf{Bool} \multimap !\mathsf{Bool}\\
\mathsf{mkFun}_R &:= \lambda s.\mathsf{let}\ s' \otimes \_ = \mathsf{for}\ j < m\ \mathsf{from}\ !\mathsf{ff} \otimes s\ \mathsf{do}\\
&\qquad \lambda p \otimes q.\mathsf{let}\ b \otimes \_ \otimes q' = (\mathsf{for}\ i < m\ \mathsf{from}\ (\mathsf{ff} \otimes q \otimes !\mathsf{ff})\ \mathsf{do}\ \mathsf{mkDep}_R[m,i,j])\ \mathsf{in}\\
&\qquad\qquad (b :: p) \otimes (\mathsf{rev}\ q')\\
&\qquad \mathsf{in}\ \mathsf{rev}\ s'\\
\mathsf{DTC}_R &:= \lambda m.\lambda n.\lambda n'.\mathsf{for}\ k < m\ \mathsf{from}\ \mathsf{ff}\ \mathsf{do}\\
&\qquad \mathsf{let}\ !x = \mathsf{It}(n', \mathsf{shift}, \mathsf{It}(k, \mathsf{mkFun}_R[m], \mathsf{It}(n, \lambda p.\mathsf{ff} :: p, \mathsf{tt} :: !\mathsf{ff})))\ \mathsf{in}\\
&\qquad \mathsf{if}\ x_0\ \mathsf{then}\ \lambda b.\mathsf{tt}\ \mathsf{else}\ \lambda b.b
\end{aligned}
$$

Figure 3 Encoding universal quantification and deterministic transitive closure.

$!\mathsf{Nat}[] \otimes A \multimap !\mathsf{Nat}[] \otimes A$ and, given $\mathsf{base} : A$, we set

$$\mathsf{for}\ i < n\ \mathsf{from}\ \mathsf{base}\ \mathsf{do}\ \mathsf{step} \quad := \quad \mathsf{It}(n, \mathsf{step}_+, !\underline{0} \otimes \mathsf{base}).$$

3.2 Expressing logspace computation

The class L of languages decidable by deterministic Turing machines with a logarithmically bounded work tape has a nice presentation in terms of descriptive complexity, due to Immerman [15]: it corresponds to first-order logic over totally ordered finite structures with the addition of a deterministic transitive closure operator. This may be equivalently presented in recursion-theoretic terms, as we do below.

We consider the following set of basic functions: the constant $0 \in \mathbb{N}$; negation $\mathsf{not} : \mathbb{B} \to \mathbb{B}$ and conjunction $\mathsf{and} : \mathbb{B}^2 \to \mathbb{B}$; the functions $\mathsf{leq} : \mathbb{N}^2 \to \mathbb{B}$, $\mathsf{sum}, \mathsf{times} : \mathbb{N}^3 \to \mathbb{B}$ corresponding to the integer relations $m \leq n$, $m + n = k$ and $m \cdot n = k$; the function $\mathsf{len} : \mathbb{W} \to \mathbb{N}$ returning the length of a string and $\mathsf{isOne} : \mathbb{W} \times \mathbb{N} \to \mathbb{B}$ s.t. $\mathsf{isOne}(w, i) = 1$ iff the i-th bit of w is 1. Now call $\mathcal{L}$ the smallest set of functions containing the above basic functions and closed by composition and the following schemata:

- **universal quantification:** if $R : \Gamma \times \mathbb{N}^2 \to \mathbb{B} \in \mathcal{L}$, then $\forall R : \Gamma \times \mathbb{N} \to \mathbb{B}$ mapping $(\gamma, m) \mapsto 1$ iff $R(\gamma, m, i) = 1$ for all $i < m$ is also in $\mathcal{L}$;
- **deterministic transitive closure:** let $R : \Gamma \times \mathbb{N}^{2k+1} \to \mathbb{B} \in \mathcal{L}$. This induces a partial map $R_* : \Gamma \times \mathbb{N}^{k+1} \to \mathbb{N}^k$ mapping $(\gamma, m, \overline{n}) \mapsto \overline{n}'$ if $\overline{n}'$ is unique s.t. $R(\gamma, m, \overline{n}, \overline{n}') = 1$, or undefined otherwise. Then, $\mathcal{L}$ also contains $\mathrm{DTC}(R) : \Gamma \times \mathbb{N}^{2k+1} \to \mathbb{B}$ mapping $(\gamma, m, \overline{n}, \overline{n}') \mapsto 1$ iff there exist $\overline{n}_0, \ldots, \overline{n}_l \in \mathbb{N}^k$, with $\overline{n}_0 = \overline{n}$, $\overline{n}_l = \overline{n}'$ and $\overline{n}_i \in \{0, \ldots, m-1\}^k$ for all $0 < i \leq l$, such that $R_*(\gamma, m, \overline{n}_i) = \overline{n}_{i+1}$ for all $0 \leq i < l$.

The class L corresponds exactly to the predicates $\mathbb{W} \to \mathbb{B}$ in $\mathcal{L}$.

We have already seen that the basic functions are representable in **PL**: they are either in Fig. 2 or were discussed in the previous section. The universal quantification schema is represented by the higher order term $\mathsf{Univ} : !(\mathsf{Nat}[] \multimap \mathsf{Nat}[] \multimap \mathsf{Bool}[]) \multimap \mathsf{Nat}[] \multimap \mathsf{Bool}$ defined in Fig. 3. The idea is the following: given $R : \mathbb{N}^2 \to \mathbb{B}$ and $m \in \mathbb{N}$, we use iteration

to build a stream of Booleans whose first m bits contain $R(m,0),\dots,R(m,m-1)$; then, we use $\mathsf{strmToW}$ to convert this into a string and we check that it consists entirely of ones.

Note that the variable $!R$ representing the relation on which universal quantification is applied is exponential because it appears free in the subterm $\lambda s.(R\,m\,k) :: s$, which is iterated. This means that, when we want to apply universal quantification to $t : \Gamma \multimap \mathsf{Nat}[] \multimap \mathsf{Nat}[] \multimap \mathsf{Bool}$ representing a function in $\mathcal{L}$, we will first have to convert it to a term of type $!\Gamma \multimap !(\mathsf{Nat}[] \multimap \mathsf{Nat}[] \multimap \mathsf{Bool})$ and then apply Univ to obtain a term of type $!\Gamma \multimap \mathsf{Nat}[] \multimap \mathsf{Bool}$. The extra modalities in $!\Gamma$ may then be removed because all types in Γ are storable (they are either Nat, Bool or Str). The same remark applies below.

Let us turn to representing $\mathrm{DTC}(R)$ with $R : \Gamma \to \mathbb{N}^{2k+1} \to \mathbb{B}$. First of all, we will restrict to the case $k = 1$. The general case may be treated by encoding a pairing function, which we omit here for briefness. Second, we observe that the particular determinization R_* of R used in the definition of DTC is inessential: we may as well define $R_*(\gamma, m, i)$ to be the smallest j such that $R(\gamma, m, i, j) = 1$, or undefined otherwise. Indeed, the important case is when R is already deterministic (*i.e.*, a partial function), in which the determinization is irrelevant. We will adopt the second definition here; it is possible to deal with the first at the expense of a more complex encoding.

The representation $\mathsf{DTC}_R : \mathsf{Nat}[] \multimap \mathsf{Nat}[] \multimap \mathsf{Nat}[] \multimap \mathsf{Bool}$ is given in Fig. 3, which we now explain. If $R : \mathbb{N}^3 \to \mathbb{B}$, computing $\mathrm{DTC}(R)(m,n,n')$ amounts to determining whether there is a path from n to n' in a graph G whose nodes are $[m] := \{0,\dots,m-1\}$ and s.t. there is an edge (n,n') iff $R_*(m,n) = n'$, so the out-degree of G is at most 1. To do this, we imagine a token traveling in G, its position being represented by a stream of type $!\mathsf{Bool}$ which is ff everywhere except where the token is. The edges of G may now be seen as a stream transformation $\varphi : !\mathsf{Bool} \multimap !\mathsf{Bool}$. Initially, the stream is tt at position n; applying φ will make the token move, and we may determine the existence of a path by checking the value at position n' after at most m applications of φ.

The idea behind the definition of φ is best explained with an example. Suppose that $m = 4$ and that the edges of G are $\{(0,1),(1,1),(3,2)\}$. Then, $\varphi = \lambda !x.\mathsf{ff} :: (\mathsf{xor}\,x_0\,x_1) :: x_3 :: \mathsf{ff} :: !x_4$. This works because the input stream $!x$ contains exactly one bit set to tt, so at most one of x_0, x_1 will be tt and exclusive-or is equivalent to disjunction. We cannot use disjunction because it is not flat. Observe by the way that the simultaneous presence of flat disjunction and flat duplication (*i.e.*, if dup had type $\mathsf{Bool} \multimap \mathsf{Bool} \otimes \mathsf{Bool}$ instead of its present type $\mathsf{Bool}[\mathsf{Bool} \otimes \mathsf{Bool}] \multimap \mathsf{Bool} \otimes \mathsf{Bool}$) would allow this solution to work for arbitrary relations (*i.e.*, graphs of arbitrary out-degree) and we would be able to compute arbitrary transitive closures, which is impossible unless $\mathsf{L} = \mathsf{NL}$.

The tricky task now is to compute φ from R. This is realized by $\mathsf{mkFun}_R : !\mathsf{Bool} \multimap !\mathsf{Bool}$, which operates by manipulating two streams p and q, the latter being initialized as the input stream s. For each $j \in [m]$, we determine its dependencies, *i.e.*, those $i \in [m]$ s.t. $R(m,i,j) = 1$. This is done by iterating over all $i \in [m]$ the term $\mathsf{mkDep}_R : C \multimap C$ (where $C := \mathsf{Bool} \otimes !\mathsf{Bool} \otimes !\mathsf{Bool}$): if $R(m,i,j) = 0$, the i-th element is saved in an auxiliary stream (it may contain the token, so we must preserve it); otherwise, we xor the current result with the i-th element and set this element to ff, so that it won't be considered later (if the token was there, it has now moved). This yields the determinization of R we defined above. At the end of this, the result is pushed to p and we start over with the (possibly) modified q (q also needs to be reversed because traversing it and pushing its elements into an auxiliary stream reversed their order). When we exit the outer loop, p contains the desired stream (but, again, in reverse order).

Finally, the term DTC_R does nothing but looping through all $0 \le k < m$ to determine

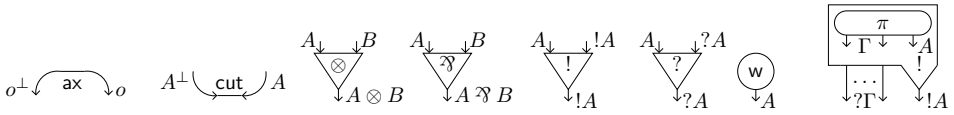


Figure 4 Cells for building nets, with their typing annotations.

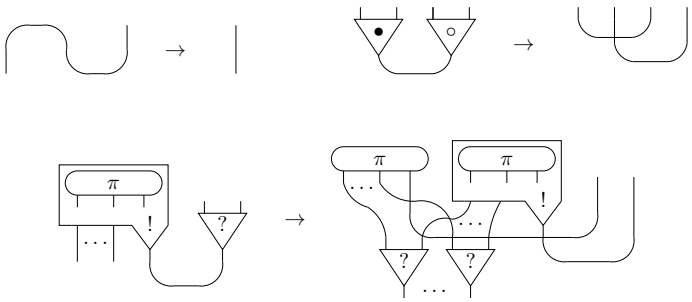


Figure 5 Cut-elimination steps on nets. On the top, $\bullet = \otimes$ and $\circ = \parr$, or $\bullet = !$ and $\circ = ?$.

whether, after k iterations of mkFun_R, the token has moved from n to n'.

4 Upper Bounds

4.1 Nets and cut-elimination (via stratification)

We will consider here *classical* simple types, generated by:

$$A, B ::= o \mid o^\perp \mid A \otimes B \mid A \parr B \mid !A \mid ?A.$$

Linear negation $A^\perp$ is defined as usual via De Morgan.

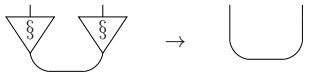
We use the standard definition of *net*, which is a labelled graph built by connecting the *cells* given in Fig. 4, respecting the orientation given therein. Each cell has a number of ports depending on the symbol it carries; the incoming are called *premises* and the outgoing conclusions. The conclusions which are not premise of any cell are called *conclusions* of the net, and so are their types. The rightmost cell in Fig. 4 is called a *box*; the conclusion labelled by $!A$ is called its *principal port*, the other are the *auxiliary ports*. A box contains a net π, whose conclusions are in bijection with the conclusions of the box itself.

The *size* of a net π is the number of its cells. Its *height* is the maximum height of its types (as trees). The *depth* of a cell or port of π is the number of nested boxes it is contained in. The *depth* of π is the maximum depth of its cells.

Nets are usually required to satisfy some form of correctness, yielding *proof nets*. We will not specify any correctness criterion here, we will rather take "proof net" as a synonym of *sequentializable net*, *i.e.*, corresponding to a sequent calculus proof (or to a typing derivation of **PL**, which will be our case).

Cut-elimination steps are defined in Fig. 5. Types (and orientations) are omitted because they can be recovered without ambiguity from Fig. 4. We prove cut-elimination for our nets using an idea of Gaboardi, Roversi and Vercelli [10] and which resorts to *stratification*.

We add the formula $\S A$ to our classical types and a corresponding cell in nets, with premise A and conclusion $\S A$, with the following cut-elimination step:



In this context, we call *plain* a net which contains no § in its types.

▶ **Definition 3** (Indexing, stratified net, level [2]). A *weak indexing* of a net π whose set of ports is P is a function $I : P \longrightarrow \mathbb{Z}$ satisfying the following:

- if p, p' are the conclusions (resp. premises) of an ax (resp. cut) cell of π, then $I(p) = I(p')$;
- if p is the conclusion of a $\otimes$ or $\parr$ cell whose premises are q, q', then $I(p) = I(q) = I(q')$;
- if p is the conclusion of a ! or ? cell whose left and right premises are l and r, then $I(p) = I(r) = I(l) - 1$;
- if p is the conclusion of a box containing π and q is the conclusion of π corresponding to p, then $I(p) = I(q) - 1$;
- if p is the conclusion of a § cell whose premise is q,then $I(p) = I(q) - 1$.

An *indexing* for π is a weak indexing I further satisfying that, for any two conclusions p, p' of π, $I(p) = I(p')$. A *stratified net* is a net admitting an indexing.

Note that, if I is a weak indexing, its range $\mathrm{rg}I$ is obviously a finite set. The *level* of I is $\ell(I) := \max \mathrm{rg}I - \min \mathrm{rg}I$.

Of course, one may easily verify that cut-elimination preserves stratification [2].

Now, the calculus we introduced in [20] (where parsimony was first introduced) is also stratified, *i.e.*, its terms may be mapped to stratified proof nets, in a way entirely analogous to the definitions we will give in Sect. 5.1 below. The polynomial-time normalization result of [20] was proved using linear explicit substitutions (in the style of [1]), which essentially amounts to using proof nets. Therefore, we have:

▶ **Proposition 4.** *Let π be a stratified proof net of size s and level l. Then, π normalizes while keeping the size of all reducts bounded by $O(s^{k(l)})$, where k depends only on l.*

Proof. The proof is similar to the usual normalization proofs for stratified systems of linear logic, such as those in [12, 2]. It actually gives a polynomial bound also on the length of the reduction and holds even in absence of types, although we will not need this. See Lemma 5 of [20]. ◀

The discovery of [10] is that simply typed nets embed in stratified nets, in a way compatible with cut-elimination.

▶ **Proposition 5** (Embedding [10]). *There is an embedding $(-)^{\S}$ of plain nets into stratified nets such that, for all π of height h:*

1. *$(\pi)^{\S}$ is of level at most h;*
2. *$\pi \to \pi'$ implies $(\pi)^{\S} \to^* (\pi')^{\S}$.*

Proof. Part 1 follows from [10, Proposition 1] and part 2 is precisely [10, Proposition 2]. The only delicate point is that those results are proved for linear logic and parsimonious logic is not a subsystem of it, because of coabsorption (the other difference is linearity vs. affinity, but it is inessential). The key observation is that indexings are oblivious to the distinction between ! and ?, so coabsorption behaves exactly as absorption, and coabsorption-free parsimonious logic *is* a subsystem of linear logic.

More precisely, given a net π, we may consider the net π^- in which all ! and ? are replaced by a self-dual modality $\sharp$, with suitable links coming from those for ! and ?, and on which indexings behave accordingly. This net will be well typed because $\sharp$ is self-dual. Then, one may check that π is stratified iff π^- is: an indexing of π induces an indexing of π^- and vice

versa. Now, it is not hard to check that, because of the above obliviousness, the embedding of [10, Proposition 1] works just as well for proof nets of the form π^-. ◀

▶ **Proposition 6.** *A plain proof net π of size s and height h normalizes while keeping the size of all reducts bounded by $O(k'(h) \cdot s^{k(h)})$, where k, k' depend solely on h.*

Proof. A consequence of point 2 of Proposition 5 and Proposition 4. The only thing to check is the size of $(\pi)^{\S}$. Let n be the maximum number of § cells added under a single axiom; the number of axioms is bounded by s, so the size of $(\pi)^{\S}$ is bounded by $(n+1)s$. But n, in turn, is bounded by h. Finally, what is the level of $(\pi)^{\S}$? Since § cells are only added to balance out the presence of exponential cells in π, the level will not exceed the depth of π, which is also bounded by h. So Proposition 4 gives us the size bound $(h+1)^{k(h)}s^{k(h)}$ (k is monotonic). ◀

4.2 Geometry of interaction

In the following definition, we use the wildcards $\bullet \in \{\otimes, ⅋\}$ and $\dagger \in \{!, ?\}$.

▶ **Definition 7** (Interaction Abstract Machine)**.** A *stack* is a finite string over $\{\mathsf{p}, \mathsf{q}\} \cup \mathbb{N}$. We use S to range over stacks and write $\alpha \cdot S$ for a stack whose first symbol is α. A stack S *matches* a formula A if: $S = \epsilon$ and $A = o$ or $A = o^{\perp}$; $S = \mathsf{m} \cdot S'$ with $\mathsf{m} \in \{\mathsf{p}, \mathsf{q}\}$, $A = A' \bullet A''$ and S' matches one of A', A''; $S = n \cdot S'$ for some $n \in \mathbb{N}$, $A = \dagger A'$ and S' matches A'.

A *box identifier* is a finite string over $\mathbb{N}$, ranged over by B. If $B = n_1 \cdots n_k$, we define $\|B\| := \max\{n_1, \ldots, n_k\}$.

Given a net π, we define an automaton $\mathsf{IAM}(\pi)$ as follows. Its *states* are tuples (d, p, B, S), where $d \in \{\uparrow, \downarrow\}$ is a *direction*, p is a port of π, B is a box identifier and S is a stack. Such a state is *admissible* if the length of B equals the depth of p and S matches the type of p. The transition relation $\rightsquigarrow$ is the smallest such that:

ax: $(\uparrow, p, B, \epsilon) \rightsquigarrow (\downarrow, p', B, \epsilon)$ whenever p, p' are the conclusions of an ax cell;
cut: $(\downarrow, p, B, S) \rightsquigarrow (\uparrow, p', B, S)$ whenever p, p' are the premises of a cut cell;
•: $(\downarrow, p, B, S) \rightsquigarrow (\downarrow, p', B, \mathsf{m} \cdot S)$ whenever p is the left (resp. right) premise of a • cell and p' its conclusion, in which case $\mathsf{m} = \mathsf{p}$ (resp. $\mathsf{m} = \mathsf{q}$);
†l: $(\downarrow, p, B, S) \rightsquigarrow (\downarrow, p', B, 0 \cdot S)$ whenever p is the left premise of a † cell and p' its conclusion;
†r: $(\downarrow, p, B, n \cdot S) \rightsquigarrow (\downarrow, p', B, (n+1) \cdot S)$ whenever p is the right premise of a † cell and p' its conclusion;
†b: $(\downarrow, p, n \cdot B, S) \rightsquigarrow (\downarrow, p', B, n \cdot S)$ whenever q is the conclusion of a box containing π and p is the conclusion of π corresponding to q;
∗: $(d'^*, p', B', S') \rightsquigarrow (d^*, p, B, S)$ whenever $(d, p, B, S) \rightsquigarrow (d', p', B', S')$, with $\uparrow^* := \downarrow$ and $\downarrow^* := \uparrow$.

Observe that the transitions are deterministic and that they preserve admissible states.

We write $\rightsquigarrow_\pi$ when we want to specify that the transition relation is that of $\mathsf{IAM}(\pi)$ (as opposed to that induced by a different net).

A sequence of transitions $s \rightsquigarrow^* s'$ of $\mathsf{IAM}(\pi)$ is *maximal* if s is admissible and the ports of s and s' are conclusions of π (not necessarily distinct). In that case, we write $s \rightsquigarrow^{\max} s'$.

The following is a standard property of the GoI. It tells us that $\mathsf{IAM}(\pi)$ behaves identically if we put π inside a box.

▶ **Lemma 8.** *Let p, p' be conclusions. Then, $(\uparrow, p, B, S) \rightsquigarrow^* (\downarrow, p', B', S')$ implies $B' = B$ and $(\uparrow, p, B_0, S) \rightsquigarrow^* (\downarrow, p', B_0, S')$ for all B_0.*

Proof. Standard. ◀

Note that cut-elimination steps preserve the number and ordering of conclusions. Hence, in the following, when $\pi \to \pi'$, we implicitly identify the conclusions of π' with those of π.

▶ **Proposition 9** (Soundness of the GoI). *Let π be a net and let $\pi \to \pi'$. Then, $\rightsquigarrow_{\pi'}^{max} = \rightsquigarrow_{\pi}^{max}$.*

Proof. The multiplicative steps are completely standard and pose no problem, so we assume that the cut-elimination step applied is exponential.

Let $s \rightsquigarrow_{\pi}^{\max} s'$. We need to show that $s \rightsquigarrow_{\pi'}^{\max} s'$. We say that the sequence crosses the left hand side of the rule if it can be decomposed into $s \rightsquigarrow_{\pi}^{*} s_1 \rightsquigarrow_{\pi}^{*} s_2 \rightsquigarrow_{\pi}^{*} s'$ such that the ports p_1, p_2 (not necessarily distinct) of the states s_1, s_2 belong to the interface of the left hand side of the rule. We will show that each crossing $s_1 \rightsquigarrow_{\pi}^{*} s_2$ induces a sequence $s_1 \rightsquigarrow_{\pi'}^{*} s_2$. This is enough to conclude, because the segments which are not crossings also exist in π' for trivial reasons (they concern subnets in which π' is identical to π).

We refer to Fig. 5, with the content of the box being renamed to ρ (since π is the proof net being reduced). Let us fix some notation. In the left hand side, we call $a_1, \dots, a_n$ the auxiliary ports and a_* the principal port of the box, and $b_1, \dots, b_n, b_*$ the associated conclusions of ρ; and q, q', r the left and right premise and the conclusion, respectively, of the ? cell. In the right hand side, we call $b'_1, \dots, b'_n, b'_*$ the conclusions of the copy of ρ which is outside the box, while the other copy still has conclusions b_i; the auxiliary ports of the box are $a'_1, \dots, a'_n$; the ports at the interface, as well as the principal port, are called as in the left hand side. We have three cases:

1. $p_1, p_2 \in \{q, q'\}$;
2. $p_1, p_2 \in \{a_1, \dots, a_n\}$;
3. $p_1 \in \{q, q'\}$ and $p_2 \in \{a_1, \dots, a_n\}$.

Actually, case 3 has a symmetric version with the roles of p_1, p_2 exchanged, but one reduces to the other thanks to the reversibility of transitions.

Case 1 works also in full linear logic. It is easy to see that we must have $p_1 = p_2$ and that everything goes well.

For case 2, let $s_1 = (\uparrow, a_i, B, n \cdot S)$. Then, we have $s_1 \rightsquigarrow_{\pi} (\uparrow, b_i, n \cdot B, S) \rightsquigarrow_{\rho}^{*} (\downarrow, b_j, n \cdot B, S') \rightsquigarrow_{\pi} (\downarrow, a_j, B, n \cdot S') = s_2$. In π', a_i becomes the conclusion of a ? link, so we have two cases: either $n = 0$, and then $s_1 \rightsquigarrow_{\pi'} (\uparrow, b'_i, B, S) \rightsquigarrow_{\rho}^{*} (\downarrow, b'_j, B, S') \rightsquigarrow_{\pi'} (\downarrow, a_j, B, 0 \cdot S') = s_2$; or $n > 0$, and then $s_1 \rightsquigarrow_{\pi'} (\uparrow, a'_i, B, (n-1) \cdot S) \rightsquigarrow_{\pi'} (\uparrow, b_i, (n-1) \cdot B, S) \rightsquigarrow_{\rho}^{*} (\downarrow, b_j, (n-1) \cdot B, S') \rightsquigarrow_{\pi'} (\downarrow, a'_j, B, (n-1) \cdot S') \rightsquigarrow_{\pi'} (\downarrow, a_j, B, n \cdot S') = s_2$. In both cases we used Lemma 8.

For case 3, suppose $p_1 = q$. We have $s_1 = (\downarrow, q, B, S) \rightsquigarrow_{\pi} (\downarrow, r, B, 0 \cdot S) \rightsquigarrow_{\pi} (\uparrow, a_*, B, 0 \cdot S) \rightsquigarrow_{\pi} (\uparrow, b_*, 0 \cdot B, S) \rightsquigarrow_{\rho}^{*} (\downarrow, b_i, 0 \cdot B, S') \rightsquigarrow_{\pi} (\downarrow, a_i, B, 0 \cdot S') = s_2$. In π', we have $s_1 \rightsquigarrow_{\pi'} (\uparrow, a'_*, B, S) \rightsquigarrow_{\rho}^{*} (\downarrow, b'_i, B, S') \rightsquigarrow_{\pi'} (\downarrow, a_i, B, 0 \cdot S') = s_2$. Suppose now $p_1 = q'$. Then, $s_1 = (\downarrow, q', B, n \cdot S) \rightsquigarrow_{\pi} (\downarrow, r, B, (n+1) \cdot S) \rightsquigarrow_{\pi} (\uparrow, a_*, B, (n+1) \cdot S) \rightsquigarrow_{\pi} (\uparrow, b_*, (n+1) \cdot B, S) \rightsquigarrow_{\rho}^{*} (\downarrow, b_i, (n+1) \cdot B, S') \rightsquigarrow_{\pi} (\downarrow, a_i, B, (n+1) \cdot S') = s_2$. In π', we have $s_1 \rightsquigarrow_{\pi'} (\uparrow, a_*, B, n \cdot S) \rightsquigarrow_{\pi'} (\uparrow, b_*, n \cdot B, S) \rightsquigarrow_{\rho}^{*} (\downarrow, b_i, n \cdot B, S') \rightsquigarrow_{\pi'} (\downarrow, a'_i, B, n \cdot S') \rightsquigarrow_{\pi'} (\downarrow, a_i, B, (n+1) \cdot S') = s_2$. We used again Lemma 8.

The above shows that $\rightsquigarrow_{\pi}^{\max} \subseteq \rightsquigarrow_{\pi'}^{\max}$. The converse is entirely analogous. ◀

▶ **Lemma 10.** *Let π be a proof net with no occurrence of ! in its conclusions, of size s and height h, let p_0 be a conclusion of π of type A containing no exponential modality and let $(\uparrow, p_0, \epsilon, S_0) \rightsquigarrow^{*} (d, p_1, B_1, S_1)$. Then, $\|B_1\| = O(k'(h) \cdot s^{k(h)})$.*

Proof. Before starting the proof, let us clarify on the restriction on A: we need it so that we have no problem in eliminating all cuts, similarly to Proposition 11 below.

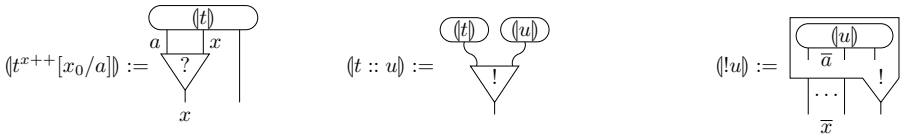
When visiting the ports of π via the execution of $\mathsf{IAM}(\pi)$, we may visit several times the same port. It is well known (cf. [5]) that the box identifier tells us which "copy" of the current port we are visiting (whence the name). For example, a box identifier of the form $3 \cdot 0 \cdot 1$ tells us that π contains three nested boxes $\mathcal{B}_1 \subseteq \mathcal{B}_2 \subseteq \mathcal{B}_3$ and that we are now in copy number 3 of $\mathcal{B}_1$, which is inside copy number 0 of $\mathcal{B}_2$, which is inside copy number 1 of $\mathcal{B}_3$ (in the usual GoI, box identifiers are more complex, in our nets they take this simple form). Therefore, in order to bound the integers appearing in box identifiers, it is enough to bound the number of times each box will be duplicated by an exponential rule while reducing π to its normal form. But this is precisely what is given by Proposition 6. ◀

5 From Simply Typed Terms to Logspace Algorithms

5.1 Translating the calculus into nets

Simple intuitionistic types are mapped to classical types in the usual way: $(\!|o|\!) := o$; $(\!|A \multimap B|\!) := (\!|A|\!)^\perp \parr (\!|B|\!)$; $(\!|A \otimes B|\!) := (\!|A|\!) \otimes (\!|B|\!)$; and $(\!|!A|\!) := !(\!|A|\!)$.

Given a type derivation of $\Gamma; \Delta \vdash t : A$, we associate with it a net of conclusions $?(\!|\Gamma|\!)^\perp, (\!|\Delta|\!)^\perp, (\!|A|\!)$. The definition is by induction on the last rule of the type derivation. The multiplicative cases are standard; η-expansion nets, defined as usual, are employed to translate the ax rule. The exponential rules (except !E, which is just a cut) are given below:



To avoid cluttering the pictures, we only drew the conclusions corresponding to those types in the context which play a role in the typing rules and we marked them with the corresponding variable. Also, types are omitted as they may be inferred from Fig. 1. In the following, we will abusively denote by $(\!|t|\!)$ the translation of a typing derivation of a term t.

The cut-elimination rules we considered are not enough to simulate reduction in the calculus. To obtain something intelligible, we must add garbage collection, *i.e.*, elimination of cuts on w cells. We define

as soon as $\pi = (\!|t|\!)$ for some term t.

▶ **Proposition 11.** *Let $\Gamma; \Delta \vdash t : A$ contain no positive (resp. negative) occurrence of ! in A (resp. Γ) and let t' be the normal form of t. Then, $(\!|t|\!) \to^* (\!|t'|\!)$ (possibly with garbage collection steps).*

Proof. The fact that cut-elimination in proof nets simulates reduction in the λ-calculus is standard, as is our translation. The type restriction here is necessary because, for conciseness, we omitted the rules reducing cuts on the auxiliary ports of boxes. ◀

We may forget about garbage collection steps, because our real interest is the following:

▶ **Corollary 12.** *Let $\Gamma; \Delta \vdash t : A$ and t' be as in Proposition 11 and let $s \rightsquigarrow^{max}_{(\!|t'|\!)} s'$. Then, we also have $s \rightsquigarrow^{max}_{(\!|t|\!)} s'$.*

Proof. We know that $(\!|t|\!) \to^* (\!|t'|\!)$ by possibly using garbage collection steps. Now, it is a standard and easy fact (see for instance [5]) that these may always be postponed, *i.e.*, we have a net π such that $(\!|t|\!) \to^* \pi \to^*_{\mathrm{gc}} (\!|t'|\!)$, where $\to^*_{\mathrm{gc}}$ denotes a reduction sequence consisting entirely of garbage collection steps while the first sequence contains none. But garbage collection does not alter maximal GoI transition sequences, because it substitutes "dead ends", *i.e.*, subnets which no maximal transition sequence may use, with shorter dead ends. Therefore, we already have $s \rightsquigarrow^{\max}_{\pi} s'$ and we may conclude by Proposition 9. ◀

5.2 Synthesis of logspace algorithms

We are at last ready to prove the inclusion $\mathsf{PL} \subseteq \mathsf{L}$. Let $t : \mathsf{Str}[A] \multimap \mathsf{Bool}$. We have to synthesize a deterministic logspace algorithm which, on input $w \in \mathbb{W}$, decides whether $t\,\underline{w} \to^* \mathsf{tt}$. In the following, every dependency (or lack thereof) is expressed w.r.t. $|w|$.

By looking at the net translation of the Boolean tt and using Corollary 12, we know that the above problem is equivalent to determining whether $(\uparrow, p, \epsilon, \mathsf{pp}) \rightsquigarrow^{\max}_{\pi} (\downarrow, p, \epsilon, \mathsf{qp})$, where p is the only conclusion of the net $\pi := (\!|t\,\underline{w}|\!)$. As observed in the introduction, the size of π is $O(|w|)$: $(\!|t|\!)$ is constant and the size of $(\!|\underline{w}|\!)$ is $c(|w|+1)+2|w|+4$, where c is a constant depending on A (it is the size of the η-expansion of $A^\perp, A$). By Lemma 10, the greatest integer that will ever appear in the B stack is bounded by $k'(h)s^{k(h)}$, where s and h are the size and height of π, respectively. This is polynomial because h is constant (it depends only on A). The depth of π is also constant: it is the maximum between the depth of $(\!|t|\!)$ (constant) and the depth of $(\!|\underline{w}|\!)$ (also constant, equal to the nesting of exponentials in A). Therefore, by the space bound given in the introduction, we may seemingly conclude.

There is however a subtlety: while we may assume $(\!|t|\!)$ to be wired into our algorithm, we still need to build $(\!|\underline{w}|\!)$ from w. Actually, instead of building the net, it will be enough to predict the behavior of $\mathsf{IAM}((\!|\underline{w}|\!))$. In fact, the only conclusion of π is a conclusion of $(\!|t|\!)$, so evaluation may start independently of w. At some point, the simulation of the automaton will reach (after crossing a cut at depth 0) a state of the form $(\uparrow, q, \epsilon, S)$, with q being where the conclusion of $(\!|\underline{w}|\!)$ should be. Our algorithm will then compute a stack S_1, according to the following cases:

1. $S = \mathsf{q}\cdot\mathsf{q}\cdot\mathsf{q}\cdot S'$: the automaton is asking for the first bit of w; if this is 1, $S_1 := \mathsf{q}\cdot\mathsf{p}\cdot 0\cdot S'$; if this is 0, $S_1 := \mathsf{p}\cdot 0\cdot S'$;
2. $S = \mathsf{q}\cdot\mathsf{q}\cdot\mathsf{p}\cdot S'$: the automaton is asking for the last bit of w; if this is 1, $S_1 := \mathsf{q}\cdot\mathsf{p}\cdot(n_1-1)\cdot S'$, where n_1 is the number of 1's in w; if this is 0, $S_1 := \mathsf{p}\cdot(n_0-1)\cdot S'$, where n_0 is the number of 0's in w;
3. $S = \mathsf{q}\cdot\mathsf{p}\cdot n\cdot\mathsf{m}\cdot S'$: the automaton is asking for the value of the neighbor of the $(n+1)$-th bit of w whose value is 1, *e.g.*, if $w = 001101$ and $n = 1$, the "second 1" is 001$\underline{1}$01, its left neighbor is 1, which is the "first 1", and its right neighbor is 0, which is the "third 0"). A request for the right (resp. left) neighbor corresponds to $m = \mathsf{p}$ (resp. $m = \mathsf{q}$). If there is no "$(n+1)$-th 1", the algorithm terminates immediately with a negative answer (the transition sequence sought in $\mathsf{IAM}(\pi)$ does not exist). Otherwise, let b be the value of the requested neighbor, and let n' be its position (as the "$(n'+1)$-th 1 or 0"). If $b = 1$, $S_1 := \mathsf{q}\cdot\mathsf{p}\cdot n'\cdot S'$; if $b = 0$ and this is the m-th zero of w, $S_1 := \mathsf{p}\cdot n'\cdot S'$;
4. $S = \mathsf{p}\cdot n\cdot\mathsf{m}\cdot S'$: the automaton is asking for the neighbor of the $(n+1)$-th bit of w whose value is 0. The same procedure as above is applied, with the roles of 0 and 1 reversed.

After computing S_1, the algorithm resumes the simulation of $\mathsf{IAM}(\pi)$ from the state $(\downarrow, q, \epsilon, S_1)$.

To understand where the four cases above come from, it is enough to look at the type of $(\!|\underline{w}|\!)$, namely $(\!|\mathsf{Str}[A]|\!) = ?(A \otimes A^\perp) \parr ?(A \otimes A^\perp) \parr A^\perp \parr A$, which is composed of four

subformulas combined by ⅋'s. The stack S must match that type; the cases correspond to the four subformulas, from right to left. The last two cases, which are similar, correspond both to $?(A \otimes A^{\perp})$; the integer and the constant m in the stack are there to match this formula. The stack S' matches A, and is returned unchanged because in $(\!|\underline{w}|\!)$ there are η-expansions of that type, acting as the identity on stacks.

The remaining details may be understood by looking at how the bits of w are represented in $(\!|\underline{w}|\!)$, but this is not really essential. What is important is to observe that predicting the behavior of $\mathsf{IAM}((\!|\underline{w}|\!))$ only requires inspecting w and updating counters bounded by $|w|$, which is all doable in logarithmic space.

6 Discussion and Perspectives

As mentioned in the introduction, we believe that our system **PL** gives the simplest functional characterization of L currently known. We also want to stress that parsimony offers a truly novel approach to applying linear logic to ICC, which is not just a variant of existing "light logics" (such as bounded, light or soft linear logic) or of systems such as those of [13, 14]. The most prominent difference with respect to "light logics" is the absence of stratification or other structural principles enforcing bounded-time cut-elimination: as mentioned above, the untyped parsimonious λ-calculus is Turing-complete, whereas light λ-calculi normalize with the same runtime independently of types. This is because parsimony is not about the global complexity of normalization but the local complexity of single reduction steps, via the notion of continuous linear approximations originally introduced in [19]. This allows dealing with non-uniform computation [20, 21], a perspective not offered by previous work on ICC.

If we add to **PL** the constant $\perp$ (typable with all types), we obtain *finitary terms* as terms whose boxes are all of the form $!\perp$. Essentially, these are purely multiplicative affine terms, or multiplicative proof nets: their size bounds the number of steps to normal form and they may be related to Boolean circuits [26]. A parsimonious term t induces a family of finitary approximations $(\lfloor t \rfloor_n)_{n\in\mathbb{N}}$: $\lfloor t \rfloor_n$ is defined by taking t and truncating all the streams appearing in it to length n ("truncating" means replacing the tail of the stream with $!\perp$). We know from [19] that reduction is continuous w.r.t. these approximations. Parsimony refines this by giving a polynomial "modulus of continuity" [20]: if $t\,\underline{w} \to^* \mathsf{b}$ with b a Boolean value, then there exists m polynomial in $|w|$ s.t. $\lfloor t \rfloor_m\,\underline{w} \to^* \mathsf{b}$, *i.e.*, a polynomial-size approximation of t is sufficient to compute the result.

Now, an arbitrary family of simply-typed finitary terms $(u_n)_{n\in\mathbb{N}} : \mathsf{Str}[] \multimap \mathsf{Bool}$ decides a language L in the same sense as a family of circuits. It is shown in [21] that, if the size of u_n is polynomial in n, then $L \in \mathsf{L/poly}$ (and conversely). If the u_n happen to be approximations of a generic **PL** term $t : \mathsf{Str}[] \multimap \mathsf{Bool}$, the family is uniform, and indeed we proved here that $L \in \mathsf{L}$. But how uniform is it? We can show that it is at least logspace-uniform, but we suspect the uniformity to be stronger (*e.g.* $\mathrm{U_E}$-uniform) and plan to investigate further on this.

Another interesting research direction is to consider second-order quantification, *i.e.*, parsimonious system **F**. In [21], it is shown that *linear* polymorphism (*i.e.*, comprehension restricted to !-free formulas) yields $\mathsf{P/poly}$ (non-uniform polynomial time). In the uniform case, we should of course obtain P, whereas we conjecture that the full parsimonious system **F** captures exactly primitive recursion.

Acknowledgments. The present formulation of this work owes much to discussions with Kazushige Terui, whom we wish to warmly thank here. We also acknowledge partial support

of ANR projects Logoi ANR-2010-BLAN-0213-02, Coquas ANR-12-JS02-006-01 and Elica ANR-14-CE25-0005.

References

1. Beniamino Accattoli and Ugo Dal Lago. Beta reduction is invariant, indeed. In *Proceedings of CSL-LICS*, page 8, 2014.
2. Patrick Baillot and Damiano Mazza. Linear logic by levels and bounded time complexity. *Theor. Comput. Sci.*, 411(2):470–503, 2010.
3. Stephen Bellantoni and Stephen A. Cook. A new recursion-theoretic characterization of the polytime functions. *Computational Complexity*, 2:97–110, 1992.
4. Guillaume Bonfante. Some programming languages for logspace and ptime. In *Proceedings of AMAST*, pages 66–80, 2006.
5. Ugo Dal Lago. Context semantics, linear logic, and computational complexity. *ACM Trans. Comput. Log.*, 10(4), 2009.
6. Ugo Dal Lago and Ulrich Schöpp. Functional programming in sublinear space. In *Proceedings of ESOP*, pages 205–225, 2010.
7. Ugo Dal Lago and Ulrich Schöpp. Type inference for sublinear space functional programming. In *Proceedings of APLAS*, pages 376–391, 2010.
8. Vincent Danos and Laurent Regnier. Reversible, irreversible and optimal lambda-machines. *Theor. Comput. Sci.*, 227(1-2):79–97, 1999.
9. Marco Gaboardi, Jean-Yves Marion, and Simona Ronchi Della Rocca. An implicit characterization of PSPACE. *ACM Trans. Comput. Log.*, 13(2):18, 2012.
10. Marco Gaboardi, Luca Roversi, and Luca Vercelli. A by-level analysis of multiplicative exponential linear logic. In *Proceedings of MFCS*, pages 344–355, 2009.
11. Jean-Yves Girard. Geometry of interaction I: Interpretation of system F. In *Proccedings of Logic Colloquium 1988*, pages 221–260, 1989.
12. Jean-Yves Girard. Light linear logic. *Inf. Comput.*, 143(2):175–204, 1998.
13. Martin Hofmann. A mixed modal/linear lambda calculus with applications to bellantoni-cook safe recursion. In *Proceedings of CSL*, pages 275–294, 1997.
14. Martin Hofmann. Linear types and non-size-increasing polynomial time computation. *Inf. Comput.*, 183(1):57–85, 2003.
15. Neil Immerman. *Descriptive complexity*. Graduate texts in computer science. Springer, 1999.
16. Neil D. Jones. Logspace and ptime characterized by programming languages. *Theor. Comput. Sci.*, 228(1-2):151–174, 1999.
17. Lars Kristiansen. Neat function algebraic characterizations of logspace and linspace. *Computational Complexity*, 14(1):72–88, 2005.
18. Daniel Leivant and Jean-Yves Marion. Lambda calculus characterizations of poly-time. *Fundam. Inform.*, 19(1/2), 1993.
19. Damiano Mazza. An infinitary affine lambda-calculus isomorphic to the full lambda-calculus. In *Proceedings of LICS*, pages 471–480, 2012.
20. Damiano Mazza. Non-uniform polytime computation in the infinitary affine lambda-calculus. In *Proceedings of ICALP, Part II*, pages 305–317, 2014.
21. Damiano Mazza and Kazushige Terui. Parsimonious types and non-uniform computation. In *Proceedings of ICALP, Part II*, pages 350–361, 2015.
22. Peter Møller Neergaard. A functional language for logarithmic space. In *Proceedings of APLAS*, pages 311–326, 2004.
23. Ramyaa Ramyaa and Daniel Leivant. Ramified corecurrence and logspace. *Electr. Notes Theor. Comput. Sci.*, 276:247–261, 2011.

24 Ulrich Schöpp. Space-efficient computation by interaction. In *Proceedings of CSL*, pages 606–621, 2006.

25 Ulrich Schöpp. Stratified bounded affine logic for logarithmic space. In *Proceedings of LICS*, pages 411–420, 2007.

26 Kazushige Terui. Proof nets and boolean circuits. In *Proceedings of LICS*, pages 182–191, 2004.

First-Order Queries on Finite Abelian Groups*

Simone Bova[1] and Barnaby Martin[2]

1 **Vienna University of Technology**
Vienna, Austria
simone.bova@tuwien.ac.at
2 **Middlesex University**
London, United Kingdom
b.martin@mdx.ac.uk

Abstract

We study the computational problem of checking whether a logical sentence is true in a finite abelian group. We prove that model checking first-order sentences on finite abelian groups is fixed-parameter tractable, when parameterized by the size of the sentence. We also prove that model checking monadic second-order sentences on finite abelian groups finitely presented by integer matrices is not fixed-parameter tractable (under standard assumptions in parameterized complexity).

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems – Computations on Discrete Structures, F.4.1 Mathematical Logic – Model theory, G.2.1 Combinatorics – Combinatorial Algorithms.

Keywords and phrases Finite Abelian Groups, First-Order Logic, Monadic Second-Order Logic

Digital Object Identifier 10.4230/LIPIcs.CSL.2015.41

1 Introduction

The *model checking* problem for first-order logic is the problem of deciding whether a given first-order sentence is true in a given finite structure; it encompasses a wide range of fundamental combinatorial problems. The problem is trivially decidable in $O(n^k)$ time, where n is the size of the structure and k is the size of the sentence, but it is not polynomial-time decidable or even *fixed-parameter tractable* when parameterized by k (under complexity assumptions in classical and parameterized complexity, respectively).

Restrictions of the model checking problem to fixed classes of structures have been intensively investigated from the perspective of parameterized algorithms and complexity. Starting from seminal work by Courcelle [4] and Seese [18], structural properties of *graphs* sufficient for fixed-parameter tractability of model checking have been identified, culminating in the recent result by Grohe, Kreutzer, and Siebertz that model checking first-order logic on classes of *nowhere dense* graphs is fixed-parameter tractable [10]. On graph classes closed under subgraphs the result is known to be tight; at the same time, there are classes of *somewhere dense* graphs (not closed under subgraphs) with fixed parameter tractable first-order (and even monadic second-order) logic model checking; the prominent examples are graph classes of bounded clique-width solved by Courcelle, Makowsky, and Rotics [5].

In contrast to its mature understanding on graphs, the model checking problem has been very little investigated on classes of structures characterized by mathematical properties,

* The first author was supported by FWF grant P26200. The second author was supported by EPSRC grant EP/L005654/1.

such as ordered structures or algebraic structures [9]. Recent work has redressed the former, ordered case [1, 2, 8], but hitherto little has been done in the latter, algebraic case.

Finite *groups* are fundamental in mathematics and computer science, and are perhaps the most prominent candidate to propose an investigation in this domain. Computational problems on finite groups are important and challenging. The notorious group isomorphism problem has long been known to be solvable in quasipolynomial time; it remains a huge open problem whether this can be improved to polynomial [16].

In contrast to general finite groups, the nice structure of finite *abelian* groups makes their associated problems simpler, both technically and computationally; isomorphism queries can be answered in linear time [13]. Yet, abelian groups remain a very important subclass; in finite model theory they appear in the literature on constraint satisfaction problems since the seminal work of Feder and Vardi [6].

Contribution. In this paper, we study the problem of model checking first-order logic on finite abelian groups. Our first contribution is a positive answer to a question posed by Grohe [9, Problem 8.2].

▶ **Result 1.** *Model checking first-order sentences on finite abelian groups, parameterized by the size of the sentence, is fixed-parameter tractable in linear time with a nonelementary parameter dependence.*

The proof is based on a revisiting of Baur-Monk's theorem on quantifier elimination in modules [11, Theorem A.1.1], which provides fresh insight into this classical result, both on important computational aspects of the class of sentences where quantifiers are eliminated, and on the mechanics the elimination procedure itself.

The theorem provides an effective procedure for reducing a first-order sentence ψ to a boolean combination of *invariant sentences* that is equivalent to ψ on abelian groups; formally, invariant sentences are first-order sentences, in the prefix class Σ_2, of the form

$$\exists x_1 \dots \exists x_k \Big(\bigwedge_{1 \leq i \leq k} \phi_1(x_i) \wedge \bigwedge_{1 \leq i < j \leq k} \neg \phi_2(x_i - x_j) \Big)$$

where $\phi_1 = \exists y_1 \dots \exists y_l \bigwedge_i \alpha_i$ and $\phi_2 = \exists z_1 \dots \exists z_m \bigwedge_j \beta_j$ are primitive positive formulas in one free variable (and k, l, and m grow with ψ).

It is unclear whether invariant sentences can be model checked in polynomial time on finite abelian groups;[1] if true, this would immediately imply a fixed-parameter tractable algorithm for model checking first-order logic on finite abelian groups. However, invariant sentences express bounds on the index of primitively positively definable subgroups (of an abelian group) into each other; for instance, the example above states that the index of the subgroup defined by $\phi_1 \wedge \phi_2$ in the subgroup defined by ϕ_1 is at least k. Therefore, if the underlying (abelian) group is finite, by Lagrange's theorem checking an invariant sentence reduces to computing the ratio between the orders of ϕ_1 and $\phi_1 \wedge \phi_2$, which is in turn the problem of counting the number of elements satisfying a primitive positive formula in one free variable in a finite abelian group.

The latter is feasible in polynomial time, and indeed in two ways: either by reducing to a linear number of calls to the algorithm by Bulatov and Dalmau for constraint satisfaction

[1] Owing to Szmielew [20], we can even assume $l = m = 1$. In this case, we can eliminate y_1 and z_1 by instantiating on all elements in the target structure, and then reduce to (a disjunction of) existentially closed conjunctions of equalities and inequalities. But this syntactic form is readily verified to be computationally hard in general.

problems on Maltsev constraints [3]; or, more directly, by reducing to a quadratic check of a formula in 2-variable logic (that is, built using only two variable symbols) using algebraic techniques.

We conclude the commentary of our first result remarking that the actual implementation of the elimination procedure, described in Section 4, is technically nontrivial, and is explicit enough to enlighten an upper bound (albeit a nonelementary one) on its complexity (which remains fairly hidden in the rather concise presentations of Baur-Monk elimination available in the literature).

In a quest to provide a measure of tightness for our first result, we investigated the problem of model checking monadic second-order logic on finite abelian groups, yet another question posed by Grohe [9]. Unfortunately we cannot answer this question, but at least we can prove the following.

▶ **Result 2.** *Model checking monadic second-order sentences on* succinctly presented *finite abelian groups, parameterized by the size of the sentence, is not fixed-parameter tractable (unless* $\mathrm{W}[1] \subseteq \mathrm{FPT}$*).*

In this setup, the group is not given as usual by its multiplication table (whose size is quadratic in the order of the group), but instead it is given by what we call a *succinct presentation*. This is a finite presentation in the usual sense [17], encoded by an integer matrix whose entries are encoded in binary as it is customary, for instance, in computational group theory and computer algebra systems. Roughly, a finite presentation is a formula of size $O(\log n)$ capable of representing a group of size n; in succinct presentations, such a representation power is already attained by formulas of size $O(\log \log n)$.

It is clear that checking formulas on structures represented succinctly is, in principle, harder. Indeed, we establish our second result by giving a fixed-parameter tractable reduction from the clique problem (parameterized by the size of the clique) to the problem of model checking monadic second-order sentences on succinctly presented finite abelian groups (parameterized by the size of the sentence).

The idea of the reduction is as follows. By the fundamental theorem [17], every finite abelian group admits a canonical decomposition as a direct sum of prime power order cyclic groups. Now, each vertex of the given graph is associated to a prime number and each edge to a positive integer; and the finite abelian group derived from the graph has a direct summand for each edge leaving each vertex (hence the direct summands are twice as much as the edges), whose prime power order is equal to the prime associated to the vertex raised to the positive integer associated to the edge (this group has a succinct presentation of linear size).

Then the key technical observation is that, despite monadic second-order logic cannot express that two sets have the same size, it can indeed express that two subsets of two cyclic subgroups of a group have the same size. Building on this, we can express by monadic second-order formulas that two direct summands of the group have the same base, or the same exponent, and therefore easily reduce a clique query on the given graph to an equivalent monadic second-order query (only depending on the size of the clique) on the derived group.

Organization. The paper is organized as follows. In Section 2, we prepare terminology and notation. In Section 3, we establish the crucial lemmas in preparation of the result on first-order logic, presented in Section 4. In Section 5, we present the result on monadic second-order logic.

2 Preliminaries

We recall some basic terminology and notation on logic, groups, and complexity, and refer the reader to any standard textbook for further details [17, 7].

For $n \geq 1$ integer, we let $[n]$ denote $\{1, \ldots, n\}$.

Logical Formulas. Throughout the paper, we work on the vocabulary $\gamma = \{+, -, 0\}$, where $+$ is a binary operation symbol, $-$ is a unary operation symbol, and 0 is a constant symbol. An *atom* has the form $t = s$ where t and s are terms built using the operation symbols in γ. We freely use the shortcut nx for the term $x + \cdots + x$ if $n > 0$, or the term $-(x + \cdots + x)$ if $n < 0$, where x occurs n times; we also write $x - y$ instead of $x + (-y)$. A *literal* is an atom or a negated atom. For every set $\{x_1, \ldots, x_l\}$ of variables, we let $\mathcal{FO}(x_1, \ldots, x_l)$ denote the class of all first-order formulas (with equality) built over γ and having free variables among $x_1, \ldots, x_l$. We let $\mathcal{FO}$ denote the class of all first-order sentences (with equality) built over γ. A first-order formula in $\mathcal{FO}(x_1, \ldots, x_l)$ is *primitive positive* if it is built from atoms using conjunction ($\wedge$) and existential quantification ($\exists$). We let $\mathcal{PP}(x_1, \ldots, x_l)$ denote the class of all primitive positive formulas in $\mathcal{FO}(x_1, \ldots, x_l)$, and $\mathcal{PP}$ denote the class of all primitive positive sentences in $\mathcal{FO}$. Similarly, for every set $\{X_1, \ldots, X_m\}$ of set variables and every set $\{x_1, \ldots, x_l\}$ of individual variables, we let $\mathcal{MSO}(X_1, \ldots, X_m, x_1, \ldots, x_l)$ denote the class of all monadic second-order formulas (with equality) built over γ and having free variables among $X_1, \ldots, X_m, x_1, \ldots, x_l$. We let $\mathcal{MSO}$ denote the class of all monadic second-order sentences (with equality) built over γ. We freely use standard shortcuts, for instance $X \subseteq Y$ instead of $(\forall x)(x \in X \to x \in Y)$, et cetera, and occasionally write

$$\begin{pmatrix} \phi_1 \\ \vdots \\ \phi_n \end{pmatrix}$$

instead of $(\phi_1 \wedge \cdots \wedge \phi_n)$.

If $\mathbb{A}$ is a structure and $\psi(X_1, \ldots, X_m, x_1, \ldots, x_l)$ is a formula, both on the same vocabulary, and f is an assignment of $X_1, \ldots, X_m$ in $\mathcal{P}(A)$ and $x_1, \ldots, x_l$ in A, we write $\mathbb{A}, f \models \psi$ if ψ is true in $\mathbb{A}$ under the assignment f. We also liberally write $\mathbb{A} \models \psi(A_1, \ldots, A_m, a_1, \ldots, a_l)$ to indicate that ψ is true in $\mathbb{A}$ under the assignment sending X_i to $A_i \in \mathcal{P}(A)$ and x_i to $a_i \in A$. Moreover, we write $\psi(X_1, \ldots, X_m, x_1, \ldots, x_l)^{\mathbb{A}}$, or $\psi^{\mathbb{A}}$ in short, to denote the set of all tuples $((A_1, \ldots, A_m), (a_1, \ldots, a_l))$ in $\mathcal{P}(A)^m \times A^l$ such that $\mathbb{A} \models \psi(A_1, \ldots, A_m, a_1, \ldots, a_l)$.

Group Theory. We view a group as a structure $\mathbb{G} = (G, +^{\mathbb{G}}, -^{\mathbb{G}}, 0^{\mathbb{G}})$ on vocabulary γ where $+^{\mathbb{G}}$ is an operation satisfying the group axioms, $0^{\mathbb{G}}$ denotes its identity element, and $-^{\mathbb{G}}g$ denotes the inverse element of $g \in G$. The group is *finite* if its order, $|G|$, is finite.

Let $\mathbb{G}$ be a group. A nonempty subset $S \subseteq G$ is (the universe of) a *subgroup* $\mathbb{S}$ of $\mathbb{G}$ if $0^{\mathbb{G}} \in S$, $-^{\mathbb{G}}s \in S$ for all $s \in S$, and $s +^{\mathbb{G}} s' \in S$ for all $s, s' \in S$. It is known that $S \subseteq G$ is a subgroup of $\mathbb{G}$ if and only if S is nonempty and $s -^{\mathbb{G}} s' \in S$ for all $s, s' \in S$; in the finite, $S \subseteq G$ is a subgroup of $\mathbb{G}$ if and only if S is nonempty and $s +^{\mathbb{G}} s' \in S$ for all $s, s' \in S$.

Let $\mathbb{G}$ be a group, let $\mathbb{S}$ be a subgroup of $\mathbb{G}$, and let $g \in G$. The *(right) coset* of $\mathbb{S}$ in $\mathbb{G}$ with respect to g, denoted by $S + g$, is the set $\{s +^{\mathbb{G}} g \colon s \in S\}$. It is known that the cosets of $\mathbb{S}$ in $\mathbb{G}$ are either identical or disjoint, and all have the same size (equal to the order of S, as S is itself a coset). Hence, the set of all cosets of $\mathbb{S}$ in $\mathbb{G}$ forms a partition of G. Consider the case where G is finite. Then, by Lagrange's theorem, the order of S divides the order of G, and $|G|/|S|$ is the number of cosets of $\mathbb{S}$ partitioning $\mathbb{G}$, known as the *index* of $\mathbb{S}$ in $\mathbb{G}$.

A group $\mathbb{G}$ is *abelian* if the operation $+^{\mathbb{G}}$ is commutative. We let $\mathcal{AG}_{\text{fin}}$ denote the class of finite abelian groups. Let $\mathbb{Z}(p,e)$ denote the cyclic group of order p^e (or equivalently the additive group modulo p^e, that is $\{0,1,\ldots,p^e-1\}$ equipped with addition modulo p^e), where p is a prime number and e a positive integer. Every finite abelian group is isomorphic to a direct sum of prime power order cyclic groups, called *primary decomposition*,

$$\mathbb{Z}(p_1,e_{1,1}) \oplus \cdots \oplus \mathbb{Z}(p_1,e_{1,n_1}) \oplus \cdots \oplus \mathbb{Z}(p_m,e_{m,1}) \oplus \cdots \oplus \mathbb{Z}(p_m,e_{m,n_m}),$$

where the p_i are prime numbers and the exponents $e_{i,j}$ are positive integers uniquely determined by the isomorphism type of the group.

A *succinct presentation* of an abelian group is a finite presentation of an abelian group encoded by an integer matrix, whose entries are encoded in binary, as customary in computational group theory. The abelian group finitely presented by the $m \times n$ integer matrix $A \in \mathbb{Z}^{m\times n}$ is the abelian group generated by the n generators $x_1,\ldots,x_n$, subject to the m relations $a_{i,1}x_1 + \ldots + a_{i,n}x_n = 0$ for $i \in [m]$. Intuitively, a binary (instead of a unary) encoding for the integer entries of the matrix corresponds to encode a term ax in size logarithmic (instead of linear) in the absolute value of a, which motivates our terminology. We let $\mathcal{AG}_{\text{spfin}}$ denote the class of all succinctly presented finite abelian groups.

Model Checking. We study the parameterized complexity of the following two computational problems. First, the problem of model checking first-order logic on finite abelian groups, in symbols $\text{MC}(\mathcal{AG}_{\text{fin}},\mathcal{FO})$, that is the problem of deciding, given $\mathbb{A} \in \mathcal{AG}_{\text{fin}}$ and $\psi \in \mathcal{FO}$, whether $\mathbb{A} \models \psi$. Second, the problem of model checking monadic second-order logic on succinctly presented finite abelian groups, in symbols $\text{MC}(\mathcal{AG}_{\text{spfin}},\mathcal{MSO})$, that is the problem of deciding, given a succinct presentation $A \in \mathbb{Z}^{m\times n}$ of a finite abelian group $\mathbb{A}$ and a sentence $\psi \in \mathcal{MSO}$, whether $\mathbb{A} \models \psi$. We regard both problems as parameterized problems, where instance $(\mathbb{A},\psi)$ is parameterized by the size of ψ.

3 Basic Facts

In this section we collect some crucial facts about the combinatorics of cosets in finite groups and about primitive positive logic over abelian groups.

We start mining, from the proof of Baur-Monk quantifier elimination theorem [11, Theorem A.1.1], a nice combinatorial property of cosets in finite groups. Roughly, in a finite group, the size of a union of cosets equals the size of the corresponding union of subgroups, hence computing the size of a union of cosets reduces to an elementary counting problem on the corresponding subgroups.

▶ **Lemma 1.** *Let $\mathbb{A}$ be a finite group. Let $\mathbb{G}$ and $\mathbb{H}_i$ $(i \in I)$ be subgroups of $\mathbb{A}$. Let C be a coset of $\mathbb{G}$ in $\mathbb{A}$ and let D_i be a coset of $\mathbb{H}_i$ in $\mathbb{A}$ $(i \in I)$. Then $C \subseteq \bigcup_{i\in I} D_i$ if and only if*

$$0 = \sum_J (-1)^{|J|} \frac{|G \cap \bigcap_{i\in J} H_i|}{|G \cap \bigcap_{i\in I} H_i|}$$

where J ranges over all subsets of I such that $C \cap \bigcap_{i\in J} D_i \neq \emptyset$.

Proof. Let $\mathbb{N}$ denote the subgroup of $\mathbb{A}$ with universe $N = G \cap \bigcap_{i\in I} H_i$. Let $C/N = \{N + c\colon c \in C\}$. In words, C/N is the set of (right) cosets of $\mathbb{N}$ in $\mathbb{A}$ with respect to elements in $C \subseteq A$. Similarly, let $D_i/N = \{N + d\colon d \in D_i\}$, $i \in I$.

Since C is a coset of $\mathbb{G}$ in $\mathbb{A}$, and $\mathbb{N}$ is a subgroup of $\mathbb{G}$, C is a (disjoint) union of cosets of $\mathbb{N}$ in $\mathbb{A}$. Similarly, D_i is a (disjoint) union of cosets of $\mathbb{N}$ in $\mathbb{A}$ ($i \in I$), and hence $\bigcup_{i\in I} D_i$ is a (disjoint) union of cosets of $\mathbb{N}$ in $\mathbb{A}$. Therefore, $C \subseteq \bigcup_{i\in I} D_i$ if and only if

$$C/N \subseteq \bigcup_{i\in I} D_i/N.$$

Since $\mathbb{A}$ is finite, C/N and D_i/N for all $i \in I$ are finite. By elementary combinatorics, if $B, B_1, \ldots, B_n$ are finite sets, then $B \subseteq \bigcup_{i\in[n]} B_i$ if and only if $0 = \sum_{I\subseteq[n]} (-1)^{|I|} |B \cap \bigcap_{i\in I} B_i|$ [12, Proposition 3.2]. Hence, $C/N \subseteq \bigcup_{i\in I} D_i/N$ if and only if

$$0 = \sum_{J\subseteq I} (-1)^{|J|} |C/N \cap \bigcap_{i\in J} D_i/N|.$$

Moreover, $C/N \cap \bigcap_{i\in J} D_i/N = (C \cap \bigcap_{i\in J} D_i)/N$ for all $J \subseteq I$, hence we reduce to

$$0 = \sum_{J\subseteq I} (-1)^{|J|} |(C \cap \bigcap_{i\in J} D_i)/N|.$$

If $C \cap \bigcap_{i\in J} D_i = \emptyset$ for some $J \subseteq I$, then the corresponding term does not contribute to the sum. Otherwise, $|(C \cap \bigcap_{i\in J} D_i)/N| = |(G \cap \bigcap_{i\in J} H_i)/N|$, and by Lagrange's theorem $|(G \cap \bigcap_{i\in J} H_i)/N| = |G \cap \bigcap_{i\in J} H_i|/|N|$, thus reducing to

$$0 = \sum_{J} (-1)^{|J|} \frac{|G \cap \bigcap_{i\in J} H_i|}{|G \cap \bigcap_{i\in I} H_i|}$$

where J ranges over all subsets of I such that $C \cap \bigcap_{i\in J} D_i \neq \emptyset$. ◀

We now make a few observations about primitive positive logic on abelian groups, starting from the folklore fact that, on abelian groups, primitive positive formulas in one free variable (respectively, with parameters) define subgroups (respectively, cosets).

▶ **Proposition 1.** *Let $\mathbb{A}$ be an abelian group.*

- *Let $\pi \in \mathcal{PP}(x)$. Then $\pi^{\mathbb{A}}$ is a subgroup of $\mathbb{A}$.*
- *Let $\pi \in \mathcal{PP}(x_1, \ldots, x_l)$ and $f\colon \{x_1, \ldots, x_{l-1}\} \to A$. If $\pi^{\mathbb{A},f} \neq \emptyset$, then $\pi^{\mathbb{A},f}$ is a coset in $\mathbb{A}$ of the subgroup $\pi(0, \ldots, 0, x_l)^{\mathbb{A}}$ of $\mathbb{A}$.*

We conclude the section describing an algorithm that, given a primitive positive formula in one free variable, returns a primitive positive formula, equivalent on abelian groups, written using only two distinct variable symbols. The algorithm is based on the computation of the Smith normal form of an integer matrix [15]; this algebraic technique is known to improve the syntactic form of primitive positive formulas [11, Lemma A.2.1], but its link with 2-variable logic is firstly and fruitfully observed here.

▶ **Proposition 2.** *There exists a single exponential time algorithm that, given a formula $\pi \in \mathcal{PP}(x)$, returns a formula $\rho \in \mathcal{PP}(x)$ of the form*

$$\rho = \bigwedge_i \exists y(c_i x = d_i y), \tag{1}$$

$c_i, d_i \in \mathbb{Z}$, such that ρ is equivalent to π on abelian groups.

Proof. Note that $\pi \in \mathcal{PP}(x)$ is equivalent on abelian groups to

$$\exists z_1 \dots \exists z_m \bigwedge_{i\in[n]} (r_i x = \sum_{j\in[m]} s_{ij} z_j)$$

where $r_i, s_{ij} \in \mathbb{Z}$, which can be displayed in matrix notation as

$$\exists z_1 \dots \exists z_m \left(R \begin{pmatrix} x \end{pmatrix} = S \begin{pmatrix} z_1 \\ \vdots \\ z_m \end{pmatrix} \right) \tag{2}$$

where $R \in \mathbb{Z}^{n\times 1}$ and $S \in \mathbb{Z}^{n\times m}$. By Smith's theorem, there exist invertible (square) matrices X and Y of orders m and n respectively such that XSY is diagonal. Therefore, upon replacing R by $XR = C$, S by $XSY = D$, and $(z_1, \dots, z_m)^T$ by $Y^{-1}(z_1, \dots, z_m)^T$, we have that (2) is equivalent on abelian groups to

$$\exists w_1 \dots \exists w_m \left(C \begin{pmatrix} x \end{pmatrix} = D \begin{pmatrix} w_1 \\ \vdots \\ w_m \end{pmatrix} \right) \tag{3}$$

where $C \in \mathbb{Z}^{n\times 1}$ and $D \in \mathbb{Z}^{n\times m}$ is diagonal.

Putting (3) back in formula notation and proceeding by logical principles, we have the following chain of equivalences on abelian groups, leading to the desired form:

$$\begin{aligned}
\pi &\equiv \exists z_1 \dots \exists z_m \bigwedge_{i\in[n]} (r_i x = \sum_{j\in[m]} s_{ij} z_j) \\
&\equiv \exists w_1 \dots \exists w_m \bigwedge_{i\in[n]} (c_i x = \sum_{j\in[m]} d_{ij} w_j) \\
&\equiv \exists w_1 \dots \exists w_m \bigwedge_{i\in[n]} (c_i x = d_{ii} w_i) \\
&\equiv \bigwedge_{i\in[n]} \exists w_i (c_i x = d_{ii} w_i) \\
&\equiv \bigwedge_{i\in[n]} \exists y (c_i x = d_{ii} y).
\end{aligned}$$

We conclude showing that ρ is computable in time single exponential in the size of π. There is an algorithm that computes D, X, and Y in time polynomial in m, n, and $s_* = \max_{i\in[m],j\in[n]} |s_{ij}|$; the integer entries in D and X have (absolute) value bounded above singly exponentially in $\max\{m, n\}$ and $\log s_*$ [19, Proposition 7.20 and Proposition 8.10]. [2] Since m, n, and s_*, as well as the entries in R, are bounded above by the size of π, it follows that the integers entries in C and D are bounded singly exponentially by the size of π. Hence ρ has size single exponential in the size of π, and is computable in time single exponential in the size of π. ◀

The nice algorithmic consequence of Proposition 2 is that we reduce the problem of computing $|\pi^{\mathbb{A}}|$, where π is a primitive positive formula on one free variable, to the problem

[2] The model of computation is an arithmetic RAM, but the algorithm translates into a polynomial-time algorithm on a standard RAM.

of computing $|\rho^{\mathbb{A}}|$, where ρ is a formula using only two variables. The latter merely requires quadratic work in the size of the structure: namely, there exists an algorithm that, given a finite abelian group $\mathbb{A}$ and a primitive positive formula $\rho(x)$ as in (1), computes the size of $\rho^{\mathbb{A}} = \{a \in A \colon \mathbb{A} \models \rho(a)\}$ in $O(k|A|^2)$ time, where k is the size of ρ.

Alternatively, for primitive positive formulas $\pi(x)$ on one free variable, it is possible to show that the problem of determining $|\pi^{\mathbb{A}}|$ is solvable in time polynomial in the size of $\mathbb{A}$ and π by calling $|A|$ times the algorithm by Bulatov and Dalmau for constraint satisfaction problems on Maltsev constraints [3]. We prefer the elementary approach of Proposition 2, as for our algorithmic result the exponential increase in size of ρ with respect to π is negligible.

4 First-Order Queries

In this section, we prove that model checking first-order logic on finite abelian groups is fixed-parameter tractable. Let $\mathbb{A}$ be a finite abelian group and let ψ be a first-order sentence in prenex form,

$$\psi = Q_1 x_1 \dots Q_m x_m \phi \tag{4}$$

where the Q_i are quantifiers, $\exists$ or $\forall$, and ϕ is a boolean combination of atoms.

We describe the algorithm referring to the pseudocode below (where $\triangleright$ denotes a comment, and $\leftarrows$ denotes an assignment). The subprocedure $\mathrm{FO}^2(\cdot)$ on Lines 7 and 10 is the algorithm in the statement of Proposition 2. The input is a pair $(\mathbb{A}, \psi)$, where $\mathbb{A}$ is a finite abelian group and ψ is a first-order sentence specified as above (Line 1).

The algorithm loops on $l = m, \dots, 1$ and constructs a first-order sentence

$$\psi_{l-1} = Q_1 x_1 \dots R_{l-1} x_{l-1} \phi_{l-1},$$

where $R_{l-1} \in \{\exists, \neg\exists\}$, such that $\mathbb{A} \models \psi_{l-1}$ if and only if $\mathbb{A} \models \psi$, and ϕ_{l-1} is a boolean combination of primitive positive formulas with free variables among $x_1, \dots, x_{l-1}$ (Lines 2-23). Intuitively, the algorithm computes ψ_{l-1} from ψ_l by "eliminating" the quantifier on variable x_l (Lines 6-14).

It follows that ψ_0 is a boolean combination, denote it by $\mathrm{bool}(\mu_1, \dots, \mu_L)$, of primitive positive sentences $\mu_1, \dots, \mu_L$ (Line 24). Moreover, $\mathbb{A} \models \psi$ if and only if $\mathbb{A} \models \psi_0$. Since each primitive positive sentence is true in $\mathbb{A}$, it holds that $\mathbb{A} \models \psi_0$ if and only if $\mathbb{A} \models \mathrm{bool}(\top, \dots, \top)$, which is easily checked (Lines 25-26).

ModelCheck$(\mathbb{A}, \psi)$
1 $\quad \triangleright\ \psi$ as in (4)
2 $\quad$ **if** $Q_m = \exists$ **then** $\psi_m \leftarrows Q_1 x_1 \dots Q_{m-1} x_{m-1} \exists x_m \mathrm{dnf}(\phi)$
3 $\quad$ **else** $\psi_m \leftarrows Q_1 x_1 \dots Q_{m-1} x_{m-1} \neg\exists x_m \mathrm{dnf}(\neg\phi)$
4 $\quad$ **for** $l = m, \dots, 1$
5 $\quad\quad \triangleright\ \psi_l = Q_1 x_1 \dots Q_{l-1} x_{l-1} R_l x_l \bigvee_{i \in I} (\pi_i \wedge \bigwedge_{j \in J_i} \neg\pi_{ij})$ where $\pi_i, \pi_{ij} \in \mathcal{PP}(x_1, \dots, x_l)$
6 $\quad\quad$ **forall** $i \in I, M \subseteq J_i, X \subseteq \mathcal{P}(M)$
7 $\quad\quad\quad \sigma_{i,M} \leftarrows \mathrm{FO}^2((\pi_i \wedge \bigwedge_{j \in M} \pi_{ij})(0, \dots, 0, x_l))$
8 $\quad\quad\quad C_{i,M} \leftarrows |\sigma_{i,M}^{\mathbb{A}}|$
9 $\quad\quad\quad$ **forall** $Y \in X$
10 $\quad\quad\quad\quad \rho_{i,Y} \leftarrows \mathrm{FO}^2((\pi_i \wedge \bigwedge_{j \in Y} \pi_{ij})(0, \dots, 0, x_l))$
11 $\quad\quad\quad\quad C_{i,Y} \leftarrows |\rho_{i,Y}^{\mathbb{A}}|$
12 $\quad\quad\quad$ **if** $0 = \sum_{Y \in X} (-1)^{|Y|} (C_{i,Y} / C_{i,M})$ **then** $\theta_{i,M,X} \leftarrows \top$ **else** $\theta_{i,M,X} \leftarrows \bot$

13 $\quad \theta_{i,M} \leftrightharpoons \bigwedge_{X \subseteq \mathcal{P}(M)} \left(\begin{pmatrix} \bigwedge_{Y \in X} \exists x_l (\pi_i \wedge \bigwedge_{j \in Y} \pi_{ij}) \\ \bigwedge_{Y \in \mathcal{P}(M) \setminus X} \neg \exists x_l (\pi_i \wedge \bigwedge_{j \in Y} \pi_{ij}) \end{pmatrix} \to \theta_{i,M,X} \right)$

14 $\quad \phi_{l-1} \leftrightharpoons \neg \bigwedge_{i \in I} \bigwedge_{M \subseteq J_i} \left(\begin{pmatrix} \exists x_l \pi_i \\ \bigwedge_{j \in M} \exists x_l \pi_{ij} \\ \bigwedge_{j \in J_i \setminus M} \neg \exists x_l \pi_{ij} \end{pmatrix} \to \theta_{i,M} \right)$

15 $\quad \triangleright$ ϕ_{l-1} boolean combination of primitive positive formulas with $x_1, \ldots, x_{l-1}$ free

16 $\quad$ **case** $Q_{l-1} = \exists, R_l = \exists$ **:**

17 $\quad\quad \psi_{l-1} \leftrightharpoons Q_1 x_1 \ldots \exists x_{l-1} \mathrm{dnf}(\phi_{l-1})$

18 $\quad$ **case** $Q_{l-1} = \exists, R_l = \neg\exists$ **:**

19 $\quad\quad \psi_{l-1} \leftrightharpoons Q_1 x_1 \ldots \exists x_{l-1} \mathrm{dnf}(\neg\phi_{l-1})$

20 $\quad$ **case** $Q_{l-1} = \forall, R_l = \exists$ **:**

21 $\quad\quad \psi_{l-1} \leftrightharpoons Q_1 x_1 \ldots \neg\exists x_{l-1} \mathrm{dnf}(\neg\phi_{l-1})$

22 $\quad$ **case** $Q_{l-1} = \forall, R_l = \neg\exists$ **:**

23 $\quad\quad \psi_{l-1} \leftrightharpoons Q_1 x_1 \ldots \neg\exists x_{l-1} \mathrm{dnf}(\phi_{l-1})$

24 $\triangleright$ $\psi_0 = \mathrm{bool}(\mu_1, \ldots, \mu_L)$ boolean combination of primitive positive sentences

25 **if** $\mathbb{A} \models \mathrm{bool}(\top, \ldots, \top)$ **then accept**

26 **reject**

We now prove that the algorithm is correct.

▶ **Lemma 2.** *Let $\mathbb{A}$ be a finite abelian group and ψ be a first-order sentence specified as in (4). Then $\mathbb{A} \models \psi$ if and only if* ModelCheck$(\mathbb{A}, \psi)$ *accepts.*

Proof. Let $\psi = Q_1 x_1 \ldots Q_m x_m \phi$, where ϕ is a boolean combination of atoms. For $l \in \{0, 1, \ldots, m\}$, let

$$\psi_l = Q_1 x_1 \ldots Q_{l-1} x_{l-1} R_l x_l \phi'_l,$$

be the formula computed by ModelCheck$(\mathbb{A}, \psi)$ either on Line 2 or 3 ($l = m$), or on Line 16, 18, 20, or 22 ($l < m$). Here, $R_l \in \{\exists, \neg\exists\}$.

By induction on $l = m, \ldots, 0$, we prove that:

(I1) $\mathbb{A} \models \psi$ if and only if $\mathbb{A} \models \psi_l$;

(I2) $\phi'_l = \bigvee_{i \in I} (\pi_i \wedge \bigwedge_{j \in J_i} \neg \pi_{ij})$, where the π_i and π_{ij} are primitive positive formulas on free variables $x_1, \ldots, x_l$.

It follows that $\mathbb{A} \models \psi$ if and only if $\mathbb{A} \models \psi_0$. Since ψ_0 is a boolean combination of primitive positive sentences, each true in $\mathbb{A}$, the correctness of the algorithm follows (Lines 24-26). We now give the inductive argument.

Base Case ($l = m$). Invariants (I1) and (I2) clearly hold if ψ_m is set as in Line 2 or 3. The operator $\mathrm{dnf}(\cdot)$, given a boolean combination of atoms, returns a logically equivalent disjunctive normal form.

Inductive Step ($l - 1$, $l \leq m$). By (I1) and (I2), we have inductively

$$\psi_l = Q_1 x_1 \ldots Q_{l-1} x_{l-1} R_l x_l \phi'_l,$$

$R_l \in \{\exists, \neg\exists\}$, such that $\mathbb{A} \models \psi$ if and only if $\mathbb{A} \models \psi_l$. Intuitively, the algorithm constructs the first-order sentence

$$\psi_{l-1} = Q_1 x_1 \ldots R_{l-1} x_{l-1} \phi'_{l-1},$$

satisfying invariants (I1) and (I2), by "eliminating" the quantifier on variable x_l in ψ_l, as follows.

Consider the case where $Q_{l-1} = R_l = \exists$ (Line 16), so that

$$\begin{aligned}\psi_l &= Q_1 x_1 \dots \exists x_{l-1} \exists x_l \phi'_l \\ &= Q_1 x_1 \dots \exists x_{l-1} \exists x_l \bigvee_{i \in I} (\pi_i \wedge \bigwedge_{j \in J_i} \neg \pi_{ij})\end{aligned}$$

where the π_i and π_{ij} are formulas in $\mathcal{PP}(x_1, \dots, x_l)$ by the induction hypothesis on ψ_l. The remaining cases (Line 18, Line 20, and Line 22) reduce to this case by handling negations as described in the pseudocode (Lines 18-23).

For readability, we first introduce the following notation. The operator $\mathcal{P}(\cdot)$, given a finite set, returns its powerset. For $i \in I$, $M \subseteq J_i$, and $X \subseteq \mathcal{P}(M)$ let:

$$\alpha_{i,M} = \exists x_l \pi_i \wedge \bigwedge_{j \in M} \exists x_l \pi_{ij} \wedge \bigwedge_{j \in J_i \setminus M} \neg \exists x_l \pi_{ij} \tag{5}$$

$$\beta_{i,M,X} = \bigwedge_{Y \in X} \exists x_l (\pi_i \wedge \bigwedge_{j \in Y} \pi_{ij}) \wedge \bigwedge_{Y \in \mathcal{P}(M) \setminus X} \neg \exists x_l (\pi_i \wedge \bigwedge_{j \in Y} \pi_{ij}) \tag{6}$$

We now claim that,

$$\exists x_l \phi'_l = \exists x_l \bigvee_{i \in I} (\pi_i \wedge \bigwedge_{j \in J_i} \neg \pi_{ij}) \tag{7}$$

$$\equiv \bigvee_{i \in I} \exists x_l (\pi_i \wedge \bigwedge_{j \in J_i} \neg \pi_{ij}) \tag{8}$$

$$\equiv \neg \bigwedge_{i \in I} \forall x_l (\pi_i \to \bigvee_{j \in J_i} \pi_{ij}) \tag{9}$$

$$\equiv \neg \bigwedge_{i \in I} \bigwedge_{M \subseteq J_i} (\alpha_{i,M} \to \forall x_l (\pi_i \to \bigvee_{j \in M} \pi_{ij})) \tag{10}$$

$$\equiv_{\mathbb{A}} \neg \bigwedge_{i \in I} \bigwedge_{M \subseteq J_i} (\alpha_{i,M} \to \bigwedge_{X \subseteq \mathcal{P}(M)} (\beta_{i,M,X} \to \theta_{i,M,X})) \tag{11}$$

$$= \phi_{l-1} \tag{12}$$

where $\theta_{i,M,X} \in \{\bot, \top\}$.

Before proving the claim, note that ϕ_{l-1} in (12) is the formula on Line 14. By the above chain of equivalences, ϕ_{l-1} is equivalent in $\mathbb{A}$ to $\exists x_l \phi'_l$. Therefore, the formula ψ_{l-1} defined on Line 17 is equivalent to ψ on $\mathbb{A}$. Hence ψ_{l-1} satisfies invariant (I1). Moreover, since $\theta_{i,M,X}$ is either $\bot$ or $\top$ (Line 12), by inspection of Lines 13 and 14 (or (5) and (6), where we observe that the variable x_l is existentially quantified in each π_i and π_{ij}), ϕ_{l-1} is a boolean combination of formulas in $\mathcal{PP}(x_1, \dots, x_{l-1})$. Therefore, the formula ψ_{l-1} defined on Line 17 by taking the disjunctive normal form of ϕ_{l-1} also satisfies invariant (I2), as desired.

We now prove the claim. The equivalences (8)-(9) hold by logical principles, and the equivalence (10) is readily verified. It remains to show that (11) holds, which is the crucial step of the construction. Here, the notation $\equiv_{\mathbb{A}}$ means that this equivalence is relative to the structure $\mathbb{A}$ (as opposed to the previous equivalences, that are logical equivalences holding for all structures).

By inspection of (11), it is sufficient to show that for all $i \in I$, $M \subseteq J_i$, and all

$f\colon \{x_1,\dots,x_{l-1}\} \to A$ such that $\mathbb{A}, f \models \alpha_{i,M}$, the following are equivalent:

$$\mathbb{A}, f \models \forall x_l(\pi_i \to \bigvee_{j\in M} \pi_{ij}) \tag{13}$$

$$\mathbb{A}, f \models \bigwedge_{X\subseteq\mathcal{P}(M)} (\beta_{i,M,X} \to \theta_{i,M,X}) \tag{14}$$

First we show that (13) is equivalent to a certain combinatorial statement involving cosets of primitive positive definable subgroups of $\mathbb{A}$, next we show that (14) is equivalent to $\mathbb{A}, f \models \theta_{i,M,X_*}$ for a suitably chosen $X \in \mathcal{P}(\mathcal{P}(M))$, and we conclude showing the equivalence of the combinatorial statement and $\mathbb{A}, f \models \theta_{i,M,X_*}$.

First, since $\mathbb{A}, f \models \alpha_{i,M}$, we have that $\pi_i^{\mathbb{A},f}$ and $\pi_{ij}^{\mathbb{A},f}$ are nonempty for all $j \in M$. Hence, by Proposition 1, $\pi_i^{\mathbb{A},f}$ is a coset in $\mathbb{A}$ of the subgroup $\pi_i(0,\dots,0,x_l)^{\mathbb{A}}$, and $\pi_{ij}^{\mathbb{A},f}$ is a coset in $\mathbb{A}$ of the subgroup $\pi_{ij}(0,\dots,0,x_l)^{\mathbb{A}}$ for all $j \in M$. We therefore have that (13) is equivalent to

$$\pi_i^{\mathbb{A},f} \subseteq \bigcup_{j\in M} \pi_{ij}^{\mathbb{A},f} \tag{15}$$

where $\pi_i^{\mathbb{A},f}$ and $\pi_{ij}^{\mathbb{A},f}$ are the described cosets in $\mathbb{A}$.

Next, observe that there exists exactly one $X \subseteq \mathcal{P}(M)$ such that $\mathbb{A}, f \models \beta_{i,M,X}$. Indeed, note that $\mathcal{P}(M)$ is partially ordered by the inclusion relation. Then the unique choice of X in $\mathcal{P}(\mathcal{P}(M))$ is determined as follows: X contains exactly those $Y \in \mathcal{P}(M)$ contained in some $Y' \in \mathcal{P}(M)$ that is maximal with the property that $\mathbb{A}, f \models \exists x_l(\pi_i \wedge \bigwedge_{j\in Y'} \pi_{ij})$. Let X_* denote this unique choice of X in $\mathcal{P}(\mathcal{P}(M))$. It follows that (14) is equivalent to

$$\mathbb{A}, f \models \theta_{i,M,X_*} \tag{16}$$

We are now in a position to conclude the argument. By Lemma 1, it holds that (15) is equivalent to

$$0 = \sum_{Y} (-1)^{|Y|} \frac{|(\pi_i \wedge \bigwedge_{j\in Y} \pi_{ij})(0,\dots,0,x_l)^{\mathbb{A}}|}{|(\pi_i \wedge \bigwedge_{j\in M} \pi_{ij})(0,\dots,0,x_l)^{\mathbb{A}}|} \tag{17}$$

where Y ranges on all subsets of M such that $(\pi_i \wedge \bigwedge_{j\in Y} \pi_{ij})^{\mathbb{A},f} \neq \emptyset$. Since $\mathbb{A}, f \models \beta_{i,M,X_*}$, it holds that X_* is exactly the set of all subsets Y of M such that $(\pi_i \wedge \bigwedge_{j\in Y} \pi_{ij})^{\mathbb{A},f} \neq \emptyset$. Hence (17) is equivalent to

$$0 = \sum_{Y\in X_*} (-1)^{|Y|} \frac{|(\pi_i \wedge \bigwedge_{j\in Y} \pi_{ij})(0,\dots,0,x_l)^{\mathbb{A}}|}{|(\pi_i \wedge \bigwedge_{j\in M} \pi_{ij})(0,\dots,0,x_l)^{\mathbb{A}}|} \tag{18}$$

By Proposition 2, the subprocedure $\mathrm{FO}^2(\cdot)$, given a primitive positive formula in one free variable, returns a primitive positive formula written using only 2 distinct variable symbols that is equivalent on abelian groups. Then, $\sigma_{i,M}$ on Line 7 is equivalent in $\mathbb{A}$ to $(\pi_i \wedge \bigwedge_{j\in M} \pi_{ij})(0,\dots,0,x_l)$, and $\rho_{i,Y}$ on Line 10 is equivalent in $\mathbb{A}$ to $(\pi_i \wedge \bigwedge_{j\in Y} \pi_{ij})(0,\dots,0,x_l)$. It follows that, on Line 8 and 11, we have that $C_{i,M} = |(\pi_i \wedge \bigwedge_{j\in M} \pi_{ij})(0,\dots,0,x_l)^{\mathbb{A}}|$ and $C_{i,Y} = |(\pi_i \wedge \bigwedge_{j\in Y} \pi_{ij})(0,\dots,0,x_l)^{\mathbb{A}}|$. Hence (18) is equivalent to

$$0 = \sum_{Y\in X_*} (-1)^{|Y|} (C_{i,Y}/C_{i,M}) \tag{19}$$

which happens exactly when θ_{i,M,X_*} is settled to $\top$ on Line 12, which is in turn equivalent to (16).

Summarizing, (13) is equivalent to (14), which settles (11), and hence the claim. The proof is complete. ◀

We analyze the runtime of the algorithm. We let $\exp_b^{i+1}(\cdot) = \exp_b(\exp_b^i(\cdot)) = b^{\exp_b^i(\cdot)}$.

▶ **Lemma 3.** *Let $\mathbb{A}$ be a finite abelian group and ψ be a first-order sentence in prenex form with m quantifiers. Then* ModelCheck$(\mathbb{A}, \psi)$ *runs in $\exp_2^{m+2}(O(k)) \cdot |A|^2$ time, where k is the size of ψ.*

Proof. For $l = m, \dots, 1$, let

$$\psi_l = Q_1 x_1 \dots Q_{l-1} x_{l-1} R_l x_l \bigvee_{i \in I_l} (\pi_i \wedge \bigwedge_{j \in J_{l,i}} \neg \pi_{ij}) \tag{20}$$

where $R_l \in \{\exists, \neg\exists\}$, and π_i and π_{ij} are in $\mathcal{PP}(x_1, \dots, x_l)$ for all $i \in I_l$ and $j \in J_{l,i}$. Note that ψ_l is the formula created on Lines 2-3 and Lines 16-23. For $l = m, \dots, 1$, we define a set $E_l \subseteq \mathcal{PP}(x_1, \dots, x_l)$ as follows

$$E_l = \{\pi_i, \pi_{ij} \colon i \in I_l, j \in J_{l,i}\},$$

and we let S_l be the size of the largest formula in E_l. We now prove by induction on $l = m, \dots, 1$ that

$$|E_l| \leq \exp_2^{m-l}(k) \tag{21}$$

$$S_l \leq k \prod_{j=l+1}^{m} |E_j| \tag{22}$$

where as usual the empty product equals 1 and $\exp_2^0(k) = k$.

The size of E_m is bounded above by the number of atoms in the sentence ψ given in input and the size of a formula in E_m is bounded above by the size of ψ, hence

$$|E_m| \leq k$$
$$S_m \leq k$$

For $l \leq m$, let $|E_l| = \exp_2^{m-l}(k)$ and $S_l = k \prod_{j=l+1}^{m} |E_j|$. Suffices to show that the following inequalities hold:

$$|E_{l-1}| \leq 2^{|E_l|}$$
$$S_{l-1} \leq |E_l| S_l$$

Indeed, the formula ϕ_{l-1} obtained in Line 15 (used to build ψ_{l-1} on Lines 16-23) is a boolean combination of the primitive positive formulas with free variables among $x_1, \dots, x_{l-1}$ created on Line 13 and 14 by existentially quantify the variable x_l in conjunctions of the form

$$\pi_i \wedge \bigwedge_{j \in S} \pi_{ij}$$

where S is a subset (of the index set) of E_l. Thus there are at most $2^{|E_l|}$ formulas in E_{l-1}. Moreover, by the same token, the size of a formula in E_{l-1} is bounded above by the number of formulas in E_l times the size S_l of the largest such formula.

We now analyze the runtime of the algorithm. Lines 2-3 are feasible in time single exponential in the size of the input sentence ψ. We claim that, for $l = m, \dots, 1$, the time spent on the corresponding iteration of the loop on Lines 4-23 is in $O(\exp_2^{m-l+3}(k) \cdot |A|^2)$. It follows that the whole loop on Lines 4-23 is feasible in $\exp_2^{m+2}(O(k)) \cdot |A|^2$ time (note that $m \leq k$), and the statement is settled.

We conclude the proof showing that, for $l = m, \dots, 1$, the corresponding iteration of the loop on Lines 4-23 is feasible in $O(\exp_2^{m-l+3}(k) \cdot |A|^2)$ time.

In view of (20), first note the following ($i \in I_l$ and $j \in J_{l,i}$):

- $|I_l| \leq 3^{|E_l|}$, as there are at most 3^c clauses on c distinct variables (here, a formula in E_l plays the role of a variable);
- $|J_{i,l}| \leq |E_l|$, because $J_{i,l}$ is a subset (of indices) of formulas in E_l.

As $M \subseteq J_{i,l}$ and $X \in \mathcal{P}(\mathcal{P}(M))$, it follows that the loop on Line 6 is executed at most

$$|I_l| \cdot |\mathcal{P}(J_{i,l})| \cdot |\mathcal{P}(\mathcal{P}(J_{i,l}))| \leq 3^{|E_l|} \cdot 2^{|E_l|} \cdot 2^{2^{|E_l|}}$$

times which is in $O(\exp_2^{m-l+2}(k))$.

For each iteration, $\mathrm{FO}^2(\cdot)$ in Line 7 requires time single exponential in the size of the formula given in input, by Proposition 2; the latter is a formula in E_l, hence its size is bounded above by $|E_{l+1}|S_{l+1}$, in turn in $O(|E_{l+1}|)$ by (22). Hence $\sigma_{i,M}$ has size single exponential in $|E_{l+1}|$. Then Line 8 requires time single exponential in $|E_{l+1}|$ and quadratic in $|A|$, by the remark following Proposition 2.

Line 9 iterates at most $2^{|E_l|}$ times as $|X| \leq |\mathcal{P}(M)| \leq |\mathcal{P}(J_{i,l})| \leq 2^{|E_l|}$. Each iteration requires as above time single exponential in $|E_{l+1}|$ and quadratic in $|A|$ on Lines 10 and 11, again by Proposition 2 and the surrounding discussion.

Line 12 sums at most $|X| \leq 2^{|E_l|}$ integer numbers not larger than $|A|$.

The formula $\theta_{i,M}$ on Line 13 has size at most $|\mathcal{P}(\mathcal{P}(M))| \leq 2^{2^{|E_l|}}$ (the size of the index set of the outermost conjunction), times $|\mathcal{P}(M)| \leq 2^{|E_l|}$ (the number of formulas on the left of the implication symbol in each conjunct), times S_l (the size of the largest such formula, as they belong in E_l). Thus $\theta_{i,M}$ has size at most $2^{2^{|E_l|}}2^{|E_l|}S_l$, which is in $O(\exp_2^{m-l+2}(k))$, and is computable in the same time.

The formula ϕ_{l-1} on Line 14 has size at most $|I_l| \leq 3^{|E_l|}$ times $|\mathcal{P}(M)| \leq 2^{|E_l|}$ (the sizes of the index sets of the two outermost conjunctions) times an upper bound on the size of the conjuncts. Each conjunct has one part on the left and one part on the right of the implication symbol. The part on the right is $\theta_{i,M}$ of size $O(\exp_2^{m-l+2}(k))$ by the above argument. The part on the left has size at most $|M| \leq |E_l|$ (the number of formulas on the left of the implication symbol in each conjunct) times S_l (the size of the largest such formula, as they belong in E_l). Hence, each conjunct has size at most $O(\exp_2^{m-l+2}(k))$. Therefore, ϕ_{l-1} has size at most $O(\exp_2^{m-l+2}(k))$, and is computable in the same time.

Line 17 (or 19, or 21, or 23) are feasible in time single exponential in the size of the formula ϕ_{l-1} on Line 14, hence in $O(\exp_2^{m-l+3}(k))$ time. Summarizing, iteration l is feasible in $O(\exp_2^{m-l+3}(k) \cdot |A|^2)$ time. ◀

As the encoding of $\mathbb{A}$ has size quadratic in $|A|$, we conclude the following.

▶ **Theorem 4.** $\mathrm{MC}(\mathcal{AG}_{\mathrm{fin}}, \mathcal{FO})$ *is fixed-parameter tractable in linear time (with a nonelementary parameter dependence).*

5 Monadic Second-Order Queries

In this section, we prove that model checking monadic second-order logic is not fixed-parameter tractable on *succinctly presented* finite abelian groups.

We proceed in two steps. First, we define a family of monadic second order formulas. Next, we use these formulas to define a suitable reduction.

In the scope of this section,

$$\mathbb{A} = \mathbb{Z}(p_1, e_{1,1}) \oplus \cdots \oplus \mathbb{Z}(p_1, e_{1,d_1}) \oplus \cdots \oplus \mathbb{Z}(p_n, e_{n,1}) \oplus \ldots \oplus \mathbb{Z}(p_n, e_{n,d_n}) \tag{23}$$

is a finite abelian group, presented by its primary decomposition, where the p_i are pairwise distinct prime numbers, and the $e_{i,j}$ are positive integers.

We now introduce a family of monadic second order formulas, and describe their meaning in $\mathbb{A}$. First, we identify subgroups of $\mathbb{A}$ as follows. Let:

$$\begin{aligned} \mathrm{Sg}(X) &\leftrightharpoons 0 \in X \wedge (\forall x, y \in X)(x + y \in X) \\ \mathrm{Sg}(X, Y) &\leftrightharpoons X \subseteq Y \wedge \mathrm{Sg}(X) \wedge \mathrm{Sg}(Y) \end{aligned}$$

The following is readily verified.

- $\mathbb{A} \models \mathrm{Sg}(S)$ if and only if $S \subseteq A$ is (the universe of) a subgroup of $\mathbb{A}$ (a nonempty subset of a *finite* group is a subgroup if and only if it is closed under the group operation).
- $\mathbb{A} \models \mathrm{Sg}(R, S)$ if and only if $R \subseteq S \subseteq A$ and R and S are (universes of) subgroups $\mathbb{R}$ and $\mathbb{S}$ of $\mathbb{A}$. It follows that $\mathbb{R}$ is a subgroup of $\mathbb{S}$.

We now identify cyclic groups and their generators in $\mathbb{A}$ as follows. Let:

$$\begin{aligned} \mathrm{Cycl}(X, x) &\leftrightharpoons (\forall Y \subseteq X)((0 \in Y \wedge (\forall y \in Y)(y + x \in Y)) \to Y = X) \\ \mathrm{Cycl}(X) &\leftrightharpoons (\exists x \in X)(\forall Y)((x \in Y \wedge \mathrm{Sg}(Y, X)) \to Y = X) \end{aligned}$$

▶ **Claim 1.** *Let $\mathbb{S}$ be a subgroup of $\mathbb{A}$ with universe $S \subseteq A$ and let $g \in S$. Then $\mathbb{A} \models \mathrm{Cycl}(S, g)$ if and only if $\mathbb{S}$ is cyclic generated by g.*

If $\mathbb{S}$ is a subgroup of $\mathbb{A}$ with universe $S \subseteq A$, it follows that $\mathbb{A} \models \mathrm{Cycl}(S)$ if and only if $\mathbb{S}$ is cyclic. Among cyclic subgroups of $\mathbb{A}$, we identify prime power order cyclic subgroups of $\mathbb{A}$ as follows. Let:

$$\mathrm{PrPow}(X) \leftrightharpoons (\forall Y, Z)((\mathrm{Sg}(Y, X) \wedge \mathrm{Sg}(Z, X)) \to (Y \subseteq Z \vee Z \subseteq Y))$$

▶ **Claim 2.** *Let $\mathbb{S}$ be a nontrivial* cyclic *subgroup of $\mathbb{A}$ with universe $S \subseteq A$. Then $\mathbb{A} \models \mathrm{PrPow}(S)$ if and only if $|S| = p^e$ for some prime number p and some positive integer e.*

Call the prime power order cyclic subgroups of $\mathbb{A}$ that do not have proper prime power order cyclic supergroups in $\mathbb{A}$ *prime terms* of $\mathbb{A}$. Let $\mathrm{PrPowCyclSg}(X) \leftrightharpoons \mathrm{Sg}(S) \wedge \mathrm{Cycl}(S) \wedge \mathrm{PrPow}(S)$ and

$$\mathrm{PrTerm}(X) \leftrightharpoons \left(\begin{array}{c} \mathrm{PrPowCyclSg}(X) \\ (\forall Y) \left(\left(\begin{array}{c} \mathrm{PrPowCyclSg}(Y) \\ X \subseteq Y \end{array} \right) \to Y = X \right) \end{array} \right)$$

By the above, it follows immediately that the $\mathbb{A} \models \mathrm{PrPowCyclSg}(S)$ if and only if $\mathbb{S}$ is a prime power order cyclic subgroup of $\mathbb{A}$, where $S \subseteq A$. Moreover, for $S \subseteq A$, it holds that $\mathbb{A} \models \mathrm{PrTerm}(S)$ if and only if $\mathbb{S}$ is a prime term of $\mathbb{A}$.

We now make a key observation. Despite monadic second order logic cannot express that two sets have the same size [14, along the lines of Proposition 7.12], indeed it can express that two subsets of two cyclic subgroups of a group have the same size. The details follow. Let:

$$\mathrm{Eq}(X, Y) = (\exists Z) \left(\begin{array}{l} (\forall x \in X)(\exists! y \in Y)(x + y \in Z) \\ (\forall y \in Y)(\exists! x \in X)(x + y \in Z) \end{array} \right)$$

▶ **Claim 3.** *Let $C \subseteq A$ and $D \subseteq A$ be subsets (of universes) of prime terms of $\mathbb{A}$.*[3] *Then $\mathbb{A} \models \mathrm{Eq}(C, D)$ if and only if $|C| = |D|$.*

Proof. For the sake of notation, let $\mathbb{T}_1$ and $\mathbb{T}_2$ be respectively the first and second term in the primary decomposition of $\mathbb{A}$, of order l and m respectively (where l and m are prime powers), and let C and D be subsets of the prime terms of $\mathbb{A}$ isomorphic to $\mathbb{T}_1$ and $\mathbb{T}_2$, respectively. Then $C = \{(c_1, 0, \dots, 0), \dots, (c_{l'}, 0, \dots, 0)\}$ and $D = \{(0, d_1, \dots, 0), \dots, (0, d_{m'}, \dots, 0)\}$, where $\{c_1, \dots, c_{l'}\} \subseteq \{0, 1, \dots, l-1\}$ and $\{d_1, \dots, d_{m'}\} \subseteq \{0, 1, \dots, m-1\}$.

Assume $|C| = |D|$, and let $b' \colon C \to D$ be a bijection. Clearly, b' is completely characterized by a bijection $b \colon \{c_1, \dots, c_{l'}\} \to \{d_1, \dots, d_{m'}\}$; in particular, $l' = m'$. Let $f(Z) = \{(c_1, b(c_1), \dots, 0), \dots, (c_{l'}, b(c_{l'}), \dots, 0)\}$. We show that

$$\mathbb{A}, f \models (\forall x \in C)(\exists! y \in D)(x + y \in Z) \wedge (\forall y \in D)(\exists! x \in C)(x + y \in Z).$$

Let $(c, 0, \dots, 0) \in C$. Then, there exists exactly one $d \in D$ such that $c +^{\mathbb{A}} d \in f(Z)$, namely $d = (c, b(c), \dots, 0)$. Similarly, let $(0, d, \dots, 0) \in D$. Then, there exists exactly one $c \in C$ such that $c +^{\mathbb{A}} d \in f(Z)$, namely $c = (b^{-1}(d), d, \dots, 0)$.

Conversely, let $B \subseteq A$ be such that

$$\mathbb{A} \models (\forall x \in C)(\exists! y \in D)(x + y \in B) \wedge (\forall y \in D)(\exists! x \in C)(x + y \in B). \tag{24}$$

Then for all $c \in C$, there exists exactly one $d \in D$, such that $c +^{\mathbb{A}} d \in B$. Let $b \colon C \to D$ be the function defined by the above condition, that is $b(c) = d$ if and only if $c +^{\mathbb{A}} d \in B$. We show that b is a bijection.

For injectivity, let $c, c' \in C$ be such that $b(c) = b(c') = d \in D$. Then, $c +^{\mathbb{A}} d \in B$ and $c' +^{\mathbb{A}} d \in B$. By (24), there exists exactly one $c'' \in C$ such that $c'' +^{\mathbb{A}} d \in B$. Hence $c = c'$.

For surjectivity, let $d \in D$. By (24), there exists $c \in C$ such that $c +^{\mathbb{A}} d \in B$. Let $b(c) = d'$. Then, by definition of b, it holds that $c +^{\mathbb{A}} d' \in B$. Hence, $c +^{\mathbb{A}} d \in B$ and $c +^{\mathbb{A}} d' \in B$. By (24), there exists exactly one $d'' \in D$ such that $c +^{\mathbb{A}} d'' \in B$. Hence $d = d'$. Then $b(c) = d$, and b is surjective. ◀

Let $\mathbb{C}$ and $\mathbb{D}$ be prime terms of $\mathbb{A}$. We conclude defining formulas that establish whether the prime power order of $\mathbb{C}$ and $\mathbb{D}$ have the same base or the same exponent. First, we deal with the base:

$$\mathrm{Base}(X, Y) \leftrightharpoons \begin{pmatrix} \mathrm{Sg}(Y, X) \wedge Y \neq \{0\} \\ (\forall Z)((\mathrm{Sg}(Z, Y) \wedge Z \neq \{0\}) \to Z = Y) \end{pmatrix}$$

$$\mathrm{EqBase}(X, Y) \leftrightharpoons (\exists X', Y') \begin{pmatrix} \mathrm{Base}(X, X') \\ \mathrm{Base}(Y, Y') \\ \mathrm{Eq}(X', Y') \end{pmatrix}$$

▶ **Claim 4.** *Let $C \subseteq A$ be the universe of a prime term $\mathbb{C}$ of $\mathbb{A}$, say isomorphic to $\mathbb{Z}(p, e)$, and let $B \subseteq C$. Then $\mathbb{A} \models \mathrm{Base}(C, B)$ if and only if B is (the universe of) the subgroup of $\mathbb{C}$ is isomorphic of $\mathbb{Z}(p)$.*

Claim 3 and Claim 4 imply the following.

▶ **Claim 5.** *Let $C, D \subseteq A$ such that $\mathbb{C}$ and $\mathbb{D}$ are distinct prime terms of $\mathbb{A}$, say isomorphic to $\mathbb{Z}(p, e)$ and $\mathbb{Z}(q, d)$ respectively. Then $\mathbb{A} \models \mathrm{EqBase}(C, D)$ if and only if $p = q$.*

[3] Along similar lines, the statement can be proved more generally for cyclic subgroups of $\mathbb{A}$ whose intersection is trivial (contains only the identity).

Finally, we deal with exponents:

$$\mathrm{Exp}(X,Y) \leftrightharpoons (\forall Z)(\mathrm{Sg}(Z,X) \to (\exists! y \in Y)\mathrm{Cycl}(Z,y))$$
$$\mathrm{EqExp}(X,Y) \leftrightharpoons (\exists X', Y') \begin{pmatrix} \mathrm{Exp}(X,X') \\ \mathrm{Exp}(Y,Y') \\ \mathrm{Eq}(X',Y') \end{pmatrix}$$

Recall that every subgroup of a cyclic subgroup is cyclic. The following is clear.

▶ **Claim 6.** *Let $C \subseteq A$ such that $\mathbb{C}$ is a prime term of $\mathbb{A}$, say isomorphic to $\mathbb{Z}(p,e)$, and let $E \subseteq C$. Then $\mathbb{A} \models \mathrm{Exp}(C,E)$ if and only if E contains exactly one generator for each (necessarily, cyclic) subgroup of $\mathbb{C}$.*

Claim 3 and Claim 6 imply the following.

▶ **Claim 7.** *Let $C, D \subseteq A$ such that $\mathbb{C}$ and $\mathbb{D}$ are distinct prime terms of $\mathbb{A}$, say isomorphic to $\mathbb{Z}(p,e)$ and $\mathbb{Z}(q,d)$ respectively. Then $\mathbb{A} \models \mathrm{EqExp}(C,D)$ if and only if $e = d$.*

We now describe the reduction. A graph $\mathbf{G} = (G, E^{\mathbf{G}})$ is a relational structure on a binary relation symbol E, where $E^{\mathbf{G}} \subseteq G^2$ is symmetric and irreflexive; we liberally view $E^{\mathbf{G}}$ as a subset of 2-element subsets of G. The clique problem, Clique, is to decide, given a graph $\mathbf{G}$ and an integer $k \geq 0$, whether $\mathbf{G}$ contains a clique on k vertices. We regard Clique as a parameterized problem, where instance $(\mathbf{G}, k)$ is parameterized by k.

We give a fixed-parameter tractable reduction from Clique to $\mathrm{MC}(\mathcal{AG}_{\mathrm{spfin}}, \mathcal{MSO})$. Let $(\mathbf{G}, k)$ be an instance of Clique. Let $\mathbf{G} = (G, E^{\mathbf{G}})$, where $G = \{v_1, \ldots, v_n\}$ and $E^{\mathbf{G}} = \{e_1, \ldots, e_m\}$. For $v_i \in G$, let $\{f_{i,1}, \ldots, f_{i,d_i}\} = \{e \in E^{\mathbf{G}} \colon v_i \in e\}$, and let $\mathrm{degree}(v_i) = d_i$ denote the degree of v_i in $\mathbf{G}$. For each $i \in [n]$ and $j \in [d_i]$, let $m(i,j) \in [m]$ be such that $f_{i,j} = e_{m(i,j)}$.

We construct an instance (A, ϕ) of $\mathrm{MC}(\mathcal{AG}_{\mathrm{spfin}}, \mathcal{MSO})$, as follows. The succinct presentation A is a (square) diagonal integer matrix of order $\sum_{i \in [n]} d_i$ defined as follows. Let $p_1, \ldots, p_n$ be the first n prime numbers.

$$A = \mathrm{diag}(p_1^{m(1,1)}, \ldots, p_1^{m(1,d_1)}, p_2^{m(2,1)}, \ldots, p_2^{m(2,d_2)}, \ldots, p_n^{m(n,1)}, \ldots, p_n^{m(n,d_n)})$$

It is readily verified that the abelian group presented by A is (finite and) isomorphic to

$$\mathbb{A} = \mathbb{Z}(p_1, m(1,1)) \oplus \cdots \oplus \mathbb{Z}(p_1, m(1,d_1)) \oplus \cdots \oplus \mathbb{Z}(p_n, m(n,1)) \oplus \ldots \oplus \mathbb{Z}(p_n, m(n,d_n))$$

▶ **Example 5.** Let $\mathbf{G} = (G, E^{\mathbf{G}})$ where $G = \{v_1, v_2, v_3, v_4\}$, $E^{\mathbf{G}} = \{e_1, e_2, e_3, e_4, e_5\}$, $e_1 = \{v_1, v_2\}$, $e_2 = \{v_1, v_4\}$, $e_3 = \{v_2, v_3\}$, $e_4 = \{v_2, v_4\}$, and $e_5 = \{v_3, v_4\}$. Then $p_1 = 2$, $p_2 = 3$, $p_3 = 5$, and $p_4 = 7$. Let $m(1,1) = 1$, $m(1,2) = 2$, $m(2,1) = 1$, $m(2,2) = 3$, $m(2,3) = 4$, $m(3,1) = 3$, $m(3,2) = 5$, $m(4,1) = 2$, $m(4,2) = 4$, and $m(4,3) = 5$. We therefore have that the succinct presentation presents the finite abelian group

$$\mathbb{Z}(2,1) \oplus \mathbb{Z}(2,2) \oplus \mathbb{Z}(3,1) \oplus \mathbb{Z}(3,3) \oplus \mathbb{Z}(3,4) \oplus \mathbb{Z}(5,3) \oplus \mathbb{Z}(5,5) \oplus \mathbb{Z}(7,2) \oplus \mathbb{Z}(7,4) \oplus \mathbb{Z}(7,5)$$

We now define a monadic second order formula ϕ, as follows. Let $K = [k] \times [k-1]$. Let $c_k \colon K \to K$ be the permutation of K uniquely determined by the following conditions:

- $c_k(i,j) = (i',j')$ if and only if $c_k(i',j') = (i,j)$, that is, c_k decomposes into $k(k-1)/2$ disjoint cycles of length 2;
- $c_k(i,j) = (j+1, i)$ for all $(i,j) \in K$ such that $1 \leq i \leq j < k$.

Note that the number of pairs $(i,j) \in K$ such that $1 \leq i \leq j < k$ is equal to $\binom{k}{2}$, the number of edges in a clique on k vertices. The following example illustrates how c_k relates to a clique on k vertices.

▶ **Example 6.** We have $c_4(1,1) = (2,1)$, $c_4(1,2) = (3,1)$, $c_4(1,3) = (4,1)$, $c_4(2,1) = (1,1)$, $c_4(2,2) = (3,2)$, $c_4(2,3) = (4,2)$, $c_4(3,1) = (1,2)$, $c_4(3,2) = (2,2)$, $c_4(3,3) = (4,3)$. c_4 decomposes into 6 disjoint cycles,

$$c_4 = ((1,1)(2,1))((1,2)(3,1))((1,3)(4,1))((2,2)(3,2))((2,3)(4,2))((3,3)(4,3)),$$

and the edges of a 4-clique on vertices $\{1,2,3,4\}$ are obtained by projecting the pairs in each cycle onto their first coordinate: $\{1,2\},\{1,3\},\{1,4\},\{2,3\},\{2,4\},\{3,4\}$.

We now define ϕ as follows:

$$\phi = (\underset{(i,j)\in K}{\exists} X_{i,j}) \left(\begin{array}{c} \bigwedge_{(i,j),(i',j')\in K,(i,j)\neq(i',j')} X_{i,j} \cap X_{i',j'} = \{0\} \\ \bigwedge_{(i,j)\in K} \mathrm{PrTerm}(X_{i,j}) \\ \bigwedge_{i\in[k]} \bigwedge_{j,j'\in[k-1]} \mathrm{EqBase}(X_{i,j}, X_{i,j'}) \\ \bigwedge_{(i,j)\in K} \mathrm{EqExp}(X_{i,j}, X_{c_k(i,j)}) \end{array} \right) \tag{25}$$

▶ **Claim 8.** $(\mathbf{G},k) \in$ Clique *if and only if* $(A,\phi) \in \mathrm{MC}(\mathcal{AG}_{\mathrm{spfin}}, \mathcal{MSO})$.

Proof. Recall that the number of pairs $(i,j) \in K = [k] \times [k-1]$ such that $1 \leq i \leq j < k$ is equal to $\binom{k}{2}$, the number of edges in a clique on k vertices.

($\Rightarrow$) Let the vertices $\{u_i : i \in [k]\} \subseteq G$ and the edges $\{f_{(i,j)} : (i,j) \in K\} \subseteq E^{\mathbf{G}}$ form a clique on k vertices in $\mathbf{G}$ such that the following holds:

- $f_{(i,j)} \cap f_{(i,j')} = u_i$,
- $f_{(i,j)} = \{u_i, c_k(i,j)_1\}$,

where $c_k(i,j)_1$ denotes the projection of $c_k(i,j)$ onto the first coordinate. For $i \in [k]$, let $n(i) \in [n]$ be such that $u_i = v_{n(i)}$, and for $(i,j) \in K$, let $m(i,j) \in [m]$ be such that $f_{(i,j)} = e_{m(i,j)}$.

For $(i,j) \in K$, let $C_{i,j}$ be (the universe of) the subgroup $\mathbb{C}_{i,j}$ of $\mathbb{A}$ satisfying the following:

- $\mathbb{C}_{i,j}$ is isomorphic to $\mathbb{Z}(p_{n(i)}, m(i,j))$;
- $\mathbb{C}_{i,j}$ has no prime power order cyclic proper supergroup in $\mathbb{A}$.

By construction such subgroup $\mathbb{C}_{i,j}$ of $\mathbb{A}$ exists and is unique, hence the above definition is sound. It is easy to verify that the family of $C_{i,j}$'s witnesses the truth of (25) in $\mathbb{A}$.

($\Leftarrow$) Let $C_{i,j} \subseteq A$ for $(i,j) \in K$ witnesses that ϕ holds in $\mathbb{A}$. Therefore, the $\mathbb{C}_{i,j}$ form a family of $\binom{k}{2}$ prime terms of $\mathbb{A}$ (by the first two lines in (25)). By the third line in (25), the $\mathbb{C}_{i,j}$'s partition into k blocks $V_1,\ldots,V_k$ such that the orders of groups in block V_l are all powers of the same prime $p_{i_l} \in \{p_1,\ldots,p_n\}$.

Let $v_{i_1},\ldots,v_{i_k} \subseteq G$ be the vertices of $\mathbf{G}$ corresponding to the primes $p_{i_1},\ldots,p_{i_k}$. We claim that there are $\binom{k}{2}$ edges between the vertices $v_{i_1},\ldots,v_{i_k}$; since $\mathbf{G}$ does not contain loops nor multiedges, it follows that the vertices $v_{i_1},\ldots,v_{i_k}$ form a clique of size k in $\mathbf{G}$.

We first observe that for all $l,l' \in [k]$, $l \neq l'$, there is an edge between v_{i_l} and $v_{i_{l'}}$. By the fourth line in (25) and the definition of c_k, we have that $V_1 \cup \cdots \cup V_k$ partitions into $\binom{k}{2}$ 2-element sets $\{\mathbb{C},\mathbb{C}'\}$ such that $\mathbb{C}$ in V_l and $\mathbb{C}'$ in $V_{l'}$ $(l,l' \in [k], l \neq l')$ such that the orders of $\mathbb{C}$ and $\mathbb{C}'$ have the same exponent. By construction, such exponent is the index of an edge between the vertices corresponding to the (prime) base of the orders of $\mathbb{C}$ and $\mathbb{C}'$. Since, by construction, the index of each edge is the exponent of exactly two prime terms of $\mathbb{A}$, distinct 2-element sets of the form above contribute distinct edges, thus contributing $\binom{k}{2}$ edges in total between vertices $v_{i_1},\ldots,v_{i_k}$. ◀

The construction of ϕ only depends on k. The complexity of constructing A is determined by:

- the time to generate the first $|G| = n$ prime numbers $p_1, \ldots, p_n$ which is roughly in $O(n^3)$ as the nth prime is bounded above by n^2 and the sieve of Eratosthenes finds all primes not larger than l in time $O(l \log \log l)$;
- the size of A, a square integer matrix of order at most $n(n-1)$ whose integer entries are bounded above by $p_n^m \leq n^{2n^2}$, thus at most n^4 entries each of size in $O(n^2 \log n)$.

Therefore (A, ϕ) is computable from $(\mathbf{G}, k)$ in time $f(k)\text{poly}(n)$ for some computable function f over the natural numbers. We thus conclude the following.

▶ **Theorem 7.** $\text{MC}(\mathcal{AG}_{\text{spfin}}, \mathcal{MSO})$ *is not fixed-parameter tractable (unless* $\text{W}[1] \subseteq \text{FPT}$*).*

6 Discussion

We proved that first-order logic is fixed-parameter tractable on finite abelian groups, and monadic second-order logic is W[1]-hard on succinctly presented finite abelian groups. What is the complexity of model checking monadic second-order logic on finitely presented abelian groups (without the succinctness condition)? On finite abelian groups?

Our work suggests some questions on general groups, reasonable in that they do not settle the isomorphism problem. For example, model checking the conjunctive positive fragment (first-order sentences on the group vocabulary built using $\forall$, $\exists$, $\wedge$, and $=$) on finite abelian groups is polynomial-time tractable; this fact can be derived from the literature or established directly by our elimination technique. How hard is model checking conjunctive positive queries on finite groups? Yet, the outstanding open question concerns the parameterized complexity of first-order (and monadic second-order) properties of finite groups.

Acknowledgments. The authors thank Carlo Toffalori for a clarification on Baur-Monk theorem.

References

1 S. Bova, R. Ganian, and S. Szeider. Model Checking Existential Logic on Partially Ordered Sets. In *CSL-LICS*, 2014.
2 S. Bova, R. Ganian, and S. Szeider. Quantified Conjunctive Queries on Partially Ordered Sets. In *IPEC*, 2014.
3 A. A. Bulatov and V. Dalmau. A Simple Algorithm for Mal'tsev Constraints. *SIAM J. Comput.*, 36(1):16–27, 2006.
4 B. Courcelle. The Monadic Second-Order Logic of Graphs. I. Recognizable Sets of Finite Graphs. *Inform. Comput.*, 85(1):12–75, 1990.
5 B. Courcelle, J. A. Makowsky, and U. Rotics. Linear Time Solvable Optimization Problems on Graphs of Bounded Clique-Width. *Theory Comput. Syst.*, 33(2):125–150, 2000.
6 T. Feder and M. Vardi. The Computational Structure of Monotone Monadic SNP and Constraint Satisfaction: A Study through Datalog and Group Theory. *SIAM J. Comput.*, 28:57–104, 1999.
7 J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2010.
8 J. Gajarský, P. Hliněný, J. Obdržálek, and S. Ordyniak. Faster Existential FO Model Checking on Posets. In *ISAAC*, 2014.
9 M. Grohe. Logic, Graphs, and Algorithms. Technical Report in *ECCC*, TR07-091, 2007.
10 M. Grohe, S. Kreutzer, and S. Siebertz. Deciding First-Order Properties of Nowhere Dense Graphs. In *STOC*, 2014.

11 W. Hodges. *Model Theory*. Cambridge University Press, 1993.
12 S. Jukna. *Extremal Combinatorics*. Springer, 2001.
13 T. Kavitha. Linear Time Algorithms for Abelian Group Isomorphism and Related Problems. *J. Comput. Syst. Sci.*, 73:986–996, 2007.
14 L. Libkin. *Elements of Finite Model Theory*. Springer, 2010.
15 C.C. MacDuffee. *The Theory of Matrices*. Dover, 2004.
16 G. L. Miller. On the $n^{\log n}$ Isomorphism Technique: A Preliminary Report. In *STOC*, 1978.
17 J. Rotman. *An Introduction to the Theory of Groups*. Springer, 1999.
18 D. Seese. Linear Time Computable Problems and First-Order Descriptions. *Mathematical Structures in Computer Science*, 6(6):505–526, 1996.
19 A. Storjohann. Algorithms for Matrix Canonical Forms. Ph. D. Thesis, ETH Zürich, 2000.
20 W. Szmielew. Elementary Properties of Abelian Groups. *Fundamenta Math.*, 41:203–271, 1955.

A Definability Dichotomy for Finite Valued CSPs

Anuj Dawar and Pengming Wang

University of Cambridge Computer Laboratory, UK
{anuj.dawar, pengming.wang}@cl.cam.ac.uk

Abstract

Finite valued constraint satisfaction problems are a formalism for describing many natural optimization problems, where constraints on the values that variables can take come with rational weights and the aim is to find an assignment of minimal cost. Thapper and Živný have recently established a complexity dichotomy for finite valued constraint languages. They show that each such language either gives rise to a polynomial-time solvable optimization problem, or to an NP-hard one, and establish a criterion to distinguish the two cases. We refine the dichotomy by showing that all optimization problems in the first class are definable in fixed-point language with counting, while all languages in the second class are not definable, even in infinitary logic with counting. The definability dichotomy is not conditional on any complexity-theoretic assumption.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes

Keywords and phrases descriptive complexity, constraint satisfaction, definability, fixed-point logic, optimization

Digital Object Identifier 10.4230/LIPIcs.CSL.2015.60

1 Introduction

Constraint Satisfaction Problems (CSPs) are a widely-used formalism for describing many problems in optimization, artificial intelligence and many other areas. The classification of CSPs according to their tractability has been a major area of theoretical research ever since Feder and Vardi [8] formulated their dichotomy conjecture. The main aim is to classify various constraint satisfaction problems as either tractable (i.e. decidable in polynomial time) or NP-hard and a number of dichotomies have been established for special cases of the CSP as well as generalizations of it. In particular, Cohen et al. [5] extend the algebraic methods that have been very successful in the classification of CSPs to what they call *soft constraints*, that is constraint problems involving optimization rather than decision problems. In this context, a recent result by Thapper and Živný [12] established a complexity dichotomy for *finite valued CSPs* (VCSPs). This is a formalism for defining optimization problems that can be expressed as sums of explicitly given rational-valued functions (a more formal definition is given in Section 2). As Thapper and Živný argue, the formalism is general enough to include a wide variety of natural optimization problems. They show that every finite valued CSP is either in P or NP-hard and provide a criterion, in terms of the existence of a definable XOR function, that determines which of the two cases holds.

In this paper we are interested in the *definability* of constraint satisfaction problems in a suitable logic. Definability in logic has been a significant tool for the study of CSPs for many years. A particular logic that has received attention in this context is Datalog, the language of inductive definitions by function-free Horn clauses. A dichotomy of definability has been established in the literature, which shows that every constraint satisfaction problem on a fixed template is either definable in Datalog or it is not definable even in the much stronger C^ω—an infinitary logic with counting. This result has not been published as such but is

24th EACSL Annual Conference on Computer Science Logic (CSL 2015).
Editor: Stephan Kreutzer; pp. 60–77

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

an immediate consequence of results in [2] where it is shown that every CSP satisfying a certain algebraic condition is not definable in C^ω, and in [3] where it is shown that those that fail to satisfy this condition have bounded width and are therefore definable in Datalog. The definability dichotomy so established does not line up with the (conjectured) complexity dichotomy as it is known that there are tractable CSPs that are not definable in Datalog.

In the context of the definability of optimization problems, one needs to distinguish three kinds of definability. In general an optimization problem asks for a *solution* (which will typically be an assignment of values from some domain D to the variables V of the instance) minimising the value of a *cost* function. This problem is standardly turned into a decision problem by including a budget b in the instance and asking if there is a solution that achieves a cost of at most b. Sentences in a logic naturally define decision problems, and in the context of definability a natural question is whether the decision problem is definable. Asking for a formula that defines an actual optimal solution may not be reasonable as such a solution may not be uniquely determined by the instance and formulas in logic are generally invariant under automorphisms of the structure on which they are interpreted. An intermediate approach is to ask for a term in the logic that defines the *cost* of an optimal solution and this is our approach in this paper.

Our main result is a definability dichotomy for finite valued CSPs. In the context of optimization problems involving numerical values, Datalog is unsuitable so we adopt as our yardstick definability in fixed-point logic with counting (FPC). This is an important logic that defines a natural and powerful proper fragment of the polynomial-time decidable properties (see [6]). It should be noted that C^ω properly extends the expressive power of FPC and therefore undefinability results for the former yield undefinability results for the latter. We establish that every finite valued CSP is either definable in FPC or undefinable in C^ω. Moreover, this dichotomy lines up exactly with the complexity dichotomy of Thapper and Živný. All the valued CSPs they determine are tractable are in fact definable in FPC, and all the ones that are NP-hard are provably not in C^ω. Unlike the complexity dichotomy, the definability dichotomy is not conditional on any complexity- theoretic assumption. Even if it were the case that P = NP, the finite valued CSPs still divide into those definable in FPC and those that are not on these same lines. It should be noted that this is a feature of the classification discovered by Thapper and Živný. They identify the tractable cases with those that can be solved using the basic linear programming (BLP) relaxation and those which have the (XOR) property, and this classification is not conditional on any complexity-theoretic assumption

The positive direction of our result builds on the recent work of Anderson et al. [1] showing that solutions to explicitly given instances of linear programming are definable in FPC. Thapper and Živný show that the optimal solutions to the tractable VCSPs can be found by solving their BLP relaxation. Thus, to establish the definability of these problems in FPC it suffices to show that the reduction to the BLP is itself definable in FPC, which we do in Section 4.

For the negative direction, we use the reductions used in [12] to establish NP-hardness of VCSPs and show that these reductions can be carried out within FPC. We start with the standard CSP form of 3-SAT, which is not definable in C^ω as a consequence of results from [2]. Details of all these reductions are presented in Section 5.

There is one issue with regard to the representation of instances of VCSPs as relational structures which we need to consider in the context of definability. An instance is defined over a language which consists of a set Γ of functions from a finite domain D to the rationals. If Γ is a finite set, it is reasonable to fix the relational signature to have a relation for each

function in Γ, and the FPC formula defining the class of VCSPs would be in this fixed relational signature. However, we can also consider the *uniform* definability of VCSP(Γ) when Γ is infinite (note that only finitely many functions from the language Γ are used in constraints in any instance). A natural way to represent this is to allow the functions themselves to be elements of the relational structure coding an instance. We can show that our dichotomy holds even under this uniform representation. For simplicity of exposition, we present the results for finite Γ and then, in Section 6, we explain how the proof can be modified to the uniform case where the functions are explicitly given as elements of the structure.

2 Background

Notation. We write $\mathbb{N}$ for the natural numbers, $\mathbb{Z}$ for the integers, $\mathbb{Q}$ for the rational numbers and $\mathbb{Q}^+$ to denote the positive rationals.

We use bars $\bar{v}$ to denote vectors. A vector over a set A indexed by a set I is a function $\bar{v} : I \to A$. We write v_a for $\bar{v}(a)$. Often, but not always, the index set I is $\{1, \ldots, d\}$, an initial segment of the natural numbers. In this case, we also write $|\bar{v}|$ for the length of $\bar{v}$, i.e. d. A matrix M over A indexed by two sets I, J is a function $M : I \times J \to A$. We use the symbol $\dot\cup$ for the disjoint union operator on sets.

If $\bar{v}$ is an I-indexed vector over A and $f : A \to B$ is a function, we write $f(\bar{v})$ to denote the I-indexed vector over B obtained by applying f componentwise to $\bar{v}$.

2.1 Valued Constraint Satisfaction

We begin with the basic definitions of valued constraint satisfaction problems. These definitions are based, with minor modifications, on the definitions given in [12].

▶ **Definition 1.** Let D be a finite domain. A *valued constraint language* Γ over D is a set of functions, where each $f \in \Gamma$ has an associated arity $m = \mathrm{ar}(f)$ and $f : D^m \to \mathbb{Q}^+$.

▶ **Definition 2.** An instance of the *valued constraint satisfaction problem (*VCSP*)* over a valued constraint language Γ is a pair $I = (V, C)$, where V is a finite set of *variables* and C is a finite set of *constraints*. Each constraint in C is a triple (σ, f, q), where $f \in \Gamma$, $\sigma \in V^{\mathrm{ar}(f)}$ and $q \in \mathbb{Q}^+$.

A *solution* to an instance I of VCSP(Γ) is an assignment $h : V \to D$ of values in D to the variables in V. The *cost* of the solution h is given by $cost_I(h) := \sum_{(\sigma,f,q)\in C} q \cdot f(h(\sigma))$. The valued constraint satisfaction problem is then to find a solution with minimal cost.

In the *decision version* of the problem, an additional threshold constant $t \in \mathbb{Q}$ is given, and the question becomes whether there is a solution h with $cost_I(h) \leq t$.

Given a valued constraint language Γ, there are certain natural closures Γ' of this set of functions for which the computational complexity of VCSP(Γ) and VCSP(Γ') coincide. The first we consider is called the *expressive power* of Γ, which consists of functions that can be defined by minimising a cost function given by a fixed VCSP(Γ)-instance I over some projection of the variables in I. The second closure of Γ we consider is under scaling and translation. Both of these are given formally in the following definition.

▶ **Definition 3.** Let Γ be a valued constraint language over D. We say that a function $f : D^m \to \mathbb{Q}$, is *expressible* in Γ, if there is some instance $I_f = (V_f, C_f) \in$ VCSP(Γ) and a tuple of distinct elements $\bar{v} = (v_1, \ldots, v_m) \in V_f^m$ such that

$$f(\bar{x}) = \min_{h \in H_{\bar{x}}} cost_{I_f}(h),$$

where $H_{\bar{x}} := \{h : V_f \to D \mid h(v_i) = x_i \ , \ 1 \leq i \leq m\}$. We then say the function f is *expressed* by the instance I_f and the tuple $\bar{v}$, and call the set of all functions that can be expressed by an instance of VCSP(Γ) the *expressive power* of Γ, denoted by $\langle\Gamma\rangle$.

Furthermore, we write $f' \equiv f$ if f' is obtained from f by *scaling* and *translation*, i.e. there are $a, b \in \mathbb{Q}, a > 0$ such that $f' = a \cdot f + b$. For a valued constraint language Γ, we write $\Gamma_{\equiv}$ to denote the set $\{f' \mid f' \equiv f \text{ for some } f \in \Gamma\}$.

The next two lemmas establish that closing Γ under these operations does not change the complexity of the corresponding problem. The first of these is implicit in the literature, and we prove a stronger version of it in Lemma 13.

▶ **Lemma 4.** *Let Γ and Γ' be valued constraint languages on domain D such that $\Gamma' \subseteq \Gamma_{\equiv}$. Then* VCSP($\Gamma'$) *is polynomial-time reducible to* VCSP(Γ).

▶ **Lemma 5** (Theorem 3.4, [5])**.** *Let Γ and Γ' be valued constraint languages on domain D such that $\Gamma' \subseteq \langle\Gamma\rangle$. Then* VCSP($\Gamma'$) *is polynomial-time reducible to* VCSP(Γ).

In the study of constraint satisfaction problems, and of structure homomorphisms more generally the *core* of a structure plays an important role. The corresponding notion for valued constraint languages is given in the following definition.

▶ **Definition 6.** We call a valued constraint language Γ over domain D a *core* if for for all $a \in D$, there is some instance $I_a \in$ VCSP(Γ) such that in every minimal cost solution over I_a, some variable is assigned a. A valued constraint language Γ' over a domain $D' \subseteq D$ is a *sub-language* of Γ if it contains exactly the functions of Γ restricted to D'. We say that Γ' is a *core of* Γ, if Γ' is a sub-language of Γ and also a core.

▶ **Lemma 7** (Lemma 2.4, [12])**.** *Let Γ' be a core of Γ. Then, $\min_h cost_I(h) = \min_h cost_{I'}(h)$ for all $I \in$ VCSP(Γ) and $I' \in$ VCSP(Γ') where I' is obtained from I by replacing each function of Γ by its restriction in Γ'.*

Finally, we consider the closure of Γ under parameterized definitions. That is, we define Γ_c, the language obtained from Γ by allowing functions that are obtained from those in Γ by fixing some parameters.

▶ **Definition 8.** Let Γ be a core over D, we denote by Γ_c the language that contains exactly those functions $f : D^m \to \mathbb{Q}$ for which there exists

- a function $g \in \Gamma$, with $g : D^n \to \mathbb{Q}$ with $n \geq m$,
- a set of indices $T_f \subseteq \{1, \ldots, n\}$,
- a mapping $s_f : \{1, \ldots, n\} \backslash T_f \to \{1, \ldots, m\}$,
- and a partial assignment $t_f : T_f \to D$,

such that f is g restricted on t_f, i.e. $f(x_1, \ldots, x_m) = g(t(1), \ldots, t(n))$, where $t(i) = t_f(i)$ if $i \in T_f$, and $t(i) = x_{s_f(i)}$ otherwise. Furthermore, we fix a mapping $\gamma : \Gamma_c \to \Gamma$ that assigns each $f \in \Gamma_c$ a function $g = \gamma(f) \in \Gamma$ with the above properties.

For example, if $g \in \Gamma$, then the function f defined by $f(x_1, x_2) := g(x_1, a, x_2)$ for some fixed $a \in D$ is in Γ_c.

2.2 Linear Programming

▶ **Definition 9.** Let $\mathbb{Q}^V$ be the rational Euclidean space indexed by a set V. A *linear optimization problem* is given by a *constraint matrix* $A \in \mathbb{Q}^{C \times V}$ and vectors $\bar{b} \in \mathbb{Q}^C, \bar{c} \in \mathbb{Q}^V$. Let $P_{A,\bar{b}} := \{\bar{x} \in \mathbb{Q}^V | A\bar{x} \leq \bar{b}\}$ be the set of *feasible solutions*. The linear optimization

problem is then to determine either that $P_{A,\bar{b}} = \emptyset$, or to find a vector $\bar{y} = \text{argmax}_{\bar{x} \in P_{A,\bar{b}}} \bar{c}^T \bar{x}$, or to determine that $\max_{\bar{x} \in P_{A,\bar{b}}} \bar{c}^T \bar{x}$ is unbounded.

We speak of the *integer linear optimization problem*, if the set of feasible solutions is instead defined as $P_{A,\bar{b}} := \{\bar{x} \in \mathbb{Z}^V | A\bar{x} \leq \bar{b}\}$.

In the decision version of the problem, an additional constant $t \in \mathbb{Q}$ is given, and the task is determine whether there exists a feasible solution $\bar{x} \in P_{A,\bar{b}}$, such that $\bar{c}^T \bar{x} \geq t$.

It is often convenient to describe the linear optimization problem $(A, \bar{b}, \bar{c})$ as a system of linear inequalities $A\bar{x} \leq \bar{b}$ along with the objective $\max_{\bar{x} \in P_{A,\bar{b}}} \bar{c}^T \bar{x}$. We may also alternatively, describe an instance with a minimization objective. It is easy to see that such a system can be converted to the standard form of Defintion 9.

Let Γ now be a valued constraint language over D, and let $I = (V, C)$ be an instance of VCSP(Γ). We associate with I the following linear optimization problem in variables $\lambda_{c,\nu}$ for each $c \in C$ with $c = (\sigma, f, q)$ and $\nu \in D^{\text{ar}(f)}$, and $\mu_{x,a}$ for each $x \in V$ and $a \in D$.

$$\min \sum_{c \in C} \sum_{\nu \in D^{\text{ar}(f)}} \lambda_{c,\nu} \cdot q \cdot f(\nu) \quad \text{where } c = (\sigma, f, q) \tag{1}$$

subject to the following constraints.

For each $c \in C$ with $c = (\sigma, f, q)$, each i with $1 \leq i \leq \text{ar}(f)$ and each $a \in D$, we have

$$\sum_{\nu \in D^{\text{ar}(f)} : \nu_i = a} \lambda_{c,\nu} \;=\; \mu_{\sigma_i, a}; \tag{2}$$

for each $x \in V$, we have

$$\sum_{a \in D} \mu_{x,a} \;=\; 1; \tag{3}$$

and for all variables $\lambda_{c,\nu}$ and $\mu_{x,a}$ we have

$$0 \leq \lambda_{c,\nu} \leq 1 \quad \text{and} \quad 0 \leq \mu_{x,a} \leq 1. \tag{4}$$

A feasible *integer* solution to the above system defines a solution $h : V \to D$ to the instance I, given by $h(x) = a$ iff $\mu_{x,a} = 1$. Equations 2 then ensure that $\lambda_{c,\nu} = 1$ for $c = (\sigma, f, q)$ just in case $h(\sigma) = \nu$. Thus, it is clear that an optimal integer solution gives us an optimal solution to I.

If we consider *rational* solutions instead of integer ones, we obtain the *basic LP-relaxation* of I, which we denote BLP(I). The following theorem characterises for which languages Γ BLP(I) has the same optimal solutions as I.

For the statement of the dichotomy result from [12], we need to introduce an additional notion. We say that the property (XOR) holds for a valued constraint language Γ over domain D if there are $a, b \in D, a \neq b$, such that $\langle\Gamma\rangle$ contains a binary function f with argmin $f = \{(a, b), (b, a)\}$.

▶ **Theorem 10** (Theorem 3.3, [12]). *Let Γ be a core over some finite domain D.*

- *Either for each instance I of* VCSP(Γ)*, the optimal solutions of I are the same as* BLP(I)*;*
- *or property* (XOR) *holds for Γ_c and* VCSP(Γ) *is* NP*-hard.*

2.3 Logic

A relational *vocabulary* (also called a *signature* or a *language*) τ is a finite sequence of relation and constant symbols $(R_1, \ldots, R_k, c_1, \ldots, c_l)$, where every relation symbol R_i has a fixed

arity $a_i \in \mathbb{N}$. A structure $\mathbf{A} = (\mathrm{dom}(\mathbf{A}), R_1^{\mathbf{A}}, \ldots, R_k^{\mathbf{A}}, c_1^{\mathbf{A}}, \ldots, c_l^{\mathbf{A}})$ over the signature τ (or a τ-*structure*) consists of a non-empty set $\mathrm{dom}(\mathbf{A})$, called the *universe* of $\mathbf{A}$, together with relations $R_i^{\mathbf{A}} \subseteq \mathrm{dom}(\mathbf{A})^{a_i}$ and constants $c_j^{\mathbf{A}} \in \mathrm{dom}(\mathbf{A})$ for each $1 \leq i \leq k$ and $1 \leq j \leq l$. Members of the set $\mathrm{dom}(\mathbf{A})$ are called the *elements* of $\mathbf{A}$ and we define the *size* of $\mathbf{A}$ to be the cardinality of its universe.

2.3.1 Fixed-point Logic with Counting

Fixed-point logic with counting (FPC) is an extension of inflationary fixed-point logic with the ability to express the cardinality of definable sets. The logic has two sorts of first-order variables: *element variables*, which range over elements of the structure on which a formula is interpreted in the usual way, and *number variables*, which range over some initial segment of the natural numbers. We write element variables with lower-case Latin letters $x, y, \ldots$ and use lower-case Greek letters $\mu, \eta, \ldots$ to denote number variables.

The atomic formulas of FPC$[\tau]$ are all formulas of the form $\mu = \eta$ or $\mu \leq \eta$, where μ, η are number variables; $s = t$ where s, t are element variables or constant symbols from τ; and $R(t_1, \ldots, t_m)$, where each t_i is either an element variable or a constant symbol and R is a relation symbol (i.e. either a symbol from τ or a relational variable) of arity m. Each relational variable of arity m has an associated type from $\{\mathrm{elem}, \mathrm{num}\}^m$. The set FPC$[\tau]$ of FPC *formulas* over τ is built up from the atomic formulas by applying an inflationary fixed-point operator $[\mathbf{ifp}_{R,\bar{x}}\phi](\bar{t})$; forming *counting terms* $\#_x\phi$, where ϕ is a formula and x an element variable; forming formulas of the kind $s = t$ and $s \leq t$ where s, t are number variables or counting terms; as well as the standard first-order operations of negation, conjunction, disjunction, universal and existential quantification. Collectively, we refer to element variables and constant symbols as *element terms*, and to number variables and counting terms as *number terms*.

For the semantics, number terms take values in $\{0, \ldots, n\}$, where $n = \mathrm{dom}(\mathbf{A})$ and element terms take values in $\mathrm{dom}(\mathbf{A})$. The semantics of atomic formulas, fixed-points and first-order operations are defined as usual (c.f., e.g., [7] for details), with comparison of number terms $\mu \leq \eta$ interpreted by comparing the corresponding integers in $\{0, \ldots, n\}$. Finally, consider a counting term of the form $\#_x\phi$, where ϕ is a formula and x an element variable. Here the intended semantics is that $\#_x\phi$ denotes the number (i.e. the element of $\{0, \ldots, n\}$) of elements that satisfy the formula ϕ. For a more detailed definition of FPC, we refer the reader to [7, 10].

We also consider C^ω—the infinitary logic with counting, and finitely many variables. We will not define it formally (the interested reader may consult [11]) but we need the following two facts about it: its expressive power properly subsumes that of FPC, and it is closed under FPC-reductions, defined below.

It is known by the Immerman-Vardi theorem [7] that fixed-point logic can express all polynomial-time properties of finite ordered structures. It follows that in FPC we can express all polynomial-time relations on the number domain. In particular, we have formulas with free number variables α, β for defining sum and product, and we simply write $\alpha + \beta$ and $\alpha \cdot \beta$ to denote these formulas. For a number term α and a non-negative integer m, we write $\alpha = m$ as short-hand for the formula that says that α is exactly m. We write $\mathrm{BIT}(\alpha, \beta)$ to denote the formula that is true just in case the β-th bit in the binary expansion of α is 1. Finally, for each constant c, we assume a formula $\mathrm{MULT}_c(W, x, y)$ which works as follows. If B is an ordered set and $W \subseteq B$ is a unary relation that codes the binary representation of an integer w, then MULT_c defines a binary relation $R \subseteq B^2$ which on the lexicographic order on B^2 defines the binary representation of $c \cdot w$.

2.3.2 Reductions

We frequently consider ways of defining one structure within another in some logic L, such as first-order logic or FPC. Consider two signatures σ and τ and a logic L. An *m-ary* L-*interpretation of* τ *in* σ is a sequence of formulae of L in vocabulary σ consisting of: (i) a formula $\delta(\bar{x})$; (ii) a formula $\varepsilon(\bar{x}, \bar{y})$; (iii) for each relation symbol $R \in \tau$ of arity k, a formula $\phi_R(\bar{x}_1, \ldots, \bar{x}_k)$; and (iv) for each constant symbol $c \in \tau$, a formula $\gamma_c(\bar{x})$, where each $\bar{x}$, $\bar{y}$ or $\bar{x}_i$ is an m-tuple of free variables. We call m the *width* of the interpretation. We say that an interpretation Θ associates a τ-structure $\mathbf{B}$ to a σ-structure $\mathbf{A}$ if there is a surjective map h from the m-tuples $\{\bar{a} \in \mathrm{dom}(\mathbf{A})^m \mid \mathbf{A} \models \delta[\bar{a}]\}$ to $\mathbf{B}$ such that:

- $h(\bar{a}_1) = h(\bar{a}_2)$ if, and only if, $\mathbf{A} \models \varepsilon[\bar{a}_1, \bar{a}_2]$;
- $R^{\mathbf{B}}(h(\bar{a}_1), \ldots, h(\bar{a}_k))$ if, and only if, $\mathbf{A} \models \phi_R[\bar{a}_1, \ldots, \bar{a}_k]$;
- $h(\bar{a}) = c^{\mathbf{B}}$ if, and only if, $\mathbf{A} \models \gamma_c[\bar{a}]$.

Note that an interpretation Θ associates a τ-structure with $\mathbf{A}$ only if ε defines an equivalence relation on $\mathrm{dom}(\mathbf{A})^m$ that is a congruence with respect to the relations defined by the formulae ϕ_R and γ_c. In such cases, however, $\mathbf{B}$ is uniquely defined up to isomorphism and we write $\Theta(\mathbf{A}) := \mathbf{B}$. Throughout this paper, we will often use interpretations where ε is simply defined as the usual equality on $\bar{a}_1$ and $\bar{a}_2$. In these instances, we omit the explicit definition of ε.

The notion of interpretations is used to define logical reductions. Let C_1 and C_2 be two classes of σ- and τ-structures respectively. We say that C_1 L-*reduces* to C_2 if there is an L-interpretation Θ of τ in σ, such that $\Theta(\mathbf{A}) \in C_2$ if and only if $\mathbf{A} \in C_1$, and we write $C_1 \leq_{\mathrm{L}} C_2$.

It is not difficult to show that formulas of FPC compose with FPC-reductions in the sense that, given an interpretation Θ of τ in σ and a τ-formula ϕ, we can define a σ-formula ϕ' such that $\mathbf{A} \models \phi'$ if, and only if, $\Theta(\mathbf{A}) \models \phi$. Moreover C^ω is closed under FPC-reductions. So if C_2 is definable in C^ω and $C_1 \leq_{\mathrm{L}} C_2$, then C_1 is also definable in C^ω.

2.3.3 Representation

In order to discuss definability of constraint satisfaction and linear programming problems, we need to fix a representation of instances of these problems as relational structures. Here, we describe the representation we use.

Numbers and Vectors. We represent an integer z as a relational structure in the following way. Let $z = s \cdot x$, with $s \in \{-1, 1\}$ being the sign of z, and $x \in \mathbb{N}$, and let $b \geq \lceil \log_2(x) \rceil$. We represent z as the structure $\mathbf{z}$ with universe $\{1, \ldots, b\}$ over the vocabulary $\tau_{\mathbb{Z}} = \{X, S, <\}$, where $<$ is interpreted the usual linear order on $\{1, \ldots, b\}$; $S^{\mathbf{z}}$ is a unary relation where $S^{\mathbf{z}} = \emptyset$ indicates that $s = 1$, and $s = -1$ otherwise; and $X^{\mathbf{z}}$ is a unary relation that encodes the bit representation of x, i.e. $X^{\mathbf{z}} = \{k \in \{1, \ldots, b\} \mid \mathrm{BIT}(x, k) = 1\}$. In a similar vein, we represent a rational number $q = s \cdot \frac{x}{d}$ by a structure $\mathbf{q}$ over the domain $\tau_{\mathbb{Q}} = \{X, D, S, <\}$, where the additional relation $D^{\mathbf{q}}$ encodes the binary representation of the denominator d in the same way as before.

In order to represent vectors and matrices over integers or rationals, we have multi-sorted universes. Let T be a non-empty set, and let v be a vector of integers indexed by T. We represent v as a structure $\mathbf{v}$ with a two-sorted universe with an index sort T, and bit sorts $\{1, \ldots, b\}$, where $b \geq \lceil \log_2(|m|) \rceil$, $m = \max_{t \in T} v_t$, over the vocabulary $(X, D, S, <)$. Now, the relation S is of arity 2, and $S^{\mathbf{v}}(t, \cdot)$ encodes the sign of the integer v_t for $t \in T$. Similarly, X is a binary relation interpreted as $X^{\mathbf{v}} = \{(t, k) \in T \times \{1, \ldots, b\} \mid \mathrm{BIT}(v_t, k) = 1\}$. In

order to represent matrices $M \in \mathbb{Z}^{T_1 \times T_2}$, indexed by two sets T_1, T_2, we allow three-sorted universes with two sorts of index sets. The generalisation to rationals carries over from the numbers case. We write τ_{vec} to denote the vocabulary for vectors over $\mathbb{Q}$ and τ_{mat} for the vocabulary for matrices over $\mathbb{Q}$.

Linear Programs. Let an instance of a linear optimization problem be given by a constraint matrix $A \in \mathbb{Q}^{C \times V}$, and vectors $\bar{b} \in \mathbb{Q}^C, \bar{c} \in \mathbb{Q}^V$ over some set of variables V and constraints C. We represent this instance in the natural way as a structure over the vocabulary $\tau_{\text{LP}} = \tau_{vec} \dot{\cup} \tau_{mat}$.

We can now state the result from [1] that we require, to the effect that there is an FPC interpretation that can define solutions to linear programs.

▶ **Theorem 11** (Theorem 11, [1]). *Let an instance $(A \in \mathbb{Q}^{C \times Q}, \bar{b} \in \mathbb{Q}^C, \bar{c} \in \mathbb{Q}^V)$ of a LP be explicitly given by a relational representation in τ_{LP}. Then, there is a* FPC*-interpretation that defines a representation of $(f \in \mathbb{Q}, \bar{v} \in \mathbb{Q}^V)$, such that $f = 1$ if and only if $\max_{\bar{x} \in P_{A,\bar{b}}} \bar{c}^T \bar{x}$ is unbounded, $\bar{v} \notin P_{A,\bar{b}}$ if and only if there is no feasible solution, and $f = 0, \bar{v} = \operatorname{argmax}_{\bar{x} \in P_{A,\bar{b}}} \bar{c}^T \bar{x}$ otherwise.*

CSPs. We next examine how instances of VCSP(Γ) for finite Γ are represented as relational structures. We return to the case of infinite Γ in Section 6.

For a fixed finite language $\Gamma = \{f_1, \ldots, f_k\}$, we represent an instance I of VCSP(Γ) as a structure $\mathbf{I} = (\text{dom}(\mathbf{I}), <, (R_f^{\mathbf{I}})_{f \in \Gamma}, W_N^{\mathbf{I}}, W_D^{\mathbf{I}})$ over the vocabulary τ_Γ. The universe $\text{dom}(\mathbf{I}) = V \dot{\cup} C \dot{\cup} B$ is a three-sorted set, consisting of variables V, constraints C, and a set B of *bit positions*. We assume that $|B|$ is at least as large as the number of bits required to represent the numerator and denominator of any rational weight occurring in I. The relation $<$ is a linear order on B. The relation $R_f^{\mathbf{I}} \subseteq V^{ar(f)} \times C$ contains (σ, c) if $c = (\sigma, f, q)$ is a constraint in I. The relations $W_N^{\mathbf{I}}, W_D^{\mathbf{I}} \subseteq C \times B$ encode the weights of the constraints: $W_N^{\mathbf{I}}(c, \beta)$ (or $W_D^{\mathbf{I}}(c, \beta)$) holds if and only if the β-th bit of the bit-representation of the numerator (or denominator, respectively) of the weight of constraint c is one. For the decision version of the VCSP, we have two additional unary relations T_N and T_D in the vocabulary which encode the binary representation of the numerator and denominator of the threshold constant of the instance.

We are now ready to define what it means to express VCSP(Γ) in a logic such as FPC. For a fixed finite langauge Γ, we say that the decision version of VCSP(Γ) is definable in a logic L if there is some $\tau_\Gamma \cup \{T_N, T_D\}$-sentence ϕ of L such that $\mathbf{I} \models \phi$ if, and only if, I is satisfiable, i.e the value of the optimal solution to I is below the given target value. For the optimization problem, we say that VCSP(Γ) is definable in FPC if there is an FPC interpretaion Θ of the vocabulary $\tau_{\mathbb{Q}}$ in τ_Γ such that for any $\mathbf{I}$, $\Theta(\mathbf{I})$ codes the value of an optimal solution for the instance I. Note that if VCSP(Γ) is definable (in FPC or C^ω) in the above sense, then so is the corresponding decision problem.

The reductions we define between VCSPs in many cases preserve the optimum value between instances. Formally, an optimum preserving L-reduction from VCSP(Γ') to VCSP(Γ) is an L-interpretation Θ from $\tau_{\Gamma'}$ to τ_Γ, such that $\mathbf{I}$ has the same optimal value as $\Theta(\mathbf{I})$. It is clear that optimum preserving reductions serve also as reductions between the corresponding decision problems.

3 Definable Reductions

An essential part of the machinery that leads to Theorem 10 is that the computational complexity of VCSP(Γ) is robust under certain changes to Γ. In other words, closing the class

of functions Γ under certain natural operations does not change the complexity of the problem. This is established by showing that the distinct problems obtained are inter-reducible under polynomial-time reductions. Our aim in this section is to show that these reductions can be expressed as interpretations in a suitable logic (in some cases first-order logic suffices, and in others we need the power of counting). These reductions are used as essential links in our proofs in Section 5.

The following lemma is analogous to Lemma 5 and shows that the reductions there can be expressed as logical interpretations.

▶ **Lemma 12.** *Let Γ and Γ' be valued constraint languages of finite size over domain D such that $\Gamma' \subseteq \langle\Gamma\rangle$. Then $\mathrm{VCSP}(\Gamma') \leq_{\mathrm{FPC}} \mathrm{VCSP}(\Gamma)$, by an optimum-preserving reduction.*

Proof. The construction of the reduction follows closely the proof of Theorem 3.4. in [5], while ensuring it is definable in FPC.

Let $I = (V, C)$ be a given instance of $\mathrm{VCSP}(\Gamma')$. We fix for each function $f \in \Gamma'$ of arity m an instance $I_f = (V_f, C_f)$ of $\mathrm{VCSP}(\Gamma)$ and a m-tuple of distinct elements $\bar{v}_f \in V_f^m$ that together express f in the sense of Definition 3. The idea is now to replace each constraint $c = (\sigma, f, q) \in C$ by a copy of I_f where the variables $v_{f1}, \ldots, v_{fm}$ in I_f are identified with $\sigma_1, \ldots, \sigma_m$, and the remaining variables are fresh. Since each I_f is an instance of $\mathrm{VCSP}(\Gamma)$, the instance $J = (U, E)$ obtained after all replacements is again an instance of $\mathrm{VCSP}(\Gamma)$. Furthermore, by Definition 3 it has the same optimal solution as I.

Formally, we define the instance $J = (U, E)$ as follows. The set of variables U consists of the variables in V plus a fresh copy of the variables in V_f for each constraint in C that uses the function f, so we can identify U with the following set.

$$U = V \mathbin{\dot\cup} \{(v, c) \mid c \in C, v \in V_f\}.$$

Each constraint $c = (\sigma, f, q) \in C$ gives rise to a set of constraints E_c, representing a copy of the constraints in C_f.

$$E_c = \{(h_c(\nu), g, q \cdot r) \mid (\nu, g, r) \in C_f\},$$

where $h_c : V_f \to U$ is defined as the mapping $h_c(v) = \sigma_i$, if $v = v_{fi}$, and $h_c(v) = (v, c)$ otherwise. The set of constraints E is then simply the union of all sets E_c.

Let $\tau_\Gamma = (<, (R_f)_{f\in\Gamma}, W_N, W_D)$ and $\tau_{\Gamma'} = (<, (R_f)_{f\in\Gamma'}, W_N, W_D)$ be the vocabularies for instances of $\mathrm{VCSP}(\Gamma)$ and $\mathrm{VCSP}(\Gamma')$ respectively. We aim to define an FPC reduction $\Theta = (\bar{\delta}, \varepsilon, \phi_<, (\phi_{R_f})_{f\in\Gamma}, \phi_{W_N}, \phi_{W_D})$ such that $\mathbf{J} = \Theta(\mathbf{I})$ corresponds to the above construction of the instance J.

Let an instance $I = (V, C)$ of $\mathrm{VCSP}(\Gamma')$ be given as a structure $\mathbf{I}$ over $\tau_{\Gamma'}$ with the three-sorted universe $\mathrm{dom}(\mathbf{I}) = V \mathbin{\dot\cup} C \mathbin{\dot\cup} B$. For each m-ary function $f \in \Gamma$ we have fixed an instance $I_f = (V_f, C_f)$ and a tuple $\bar{v}_f = (v_{f1}, \ldots, v_{fm})$ that together express f. As the construction of $\mathbf{J}$ depends on these instances, we fix an encoding of them in an initial segment of the natural numbers. To be precise, as the sets $\hat{V} = \bigcup_{f\in\Gamma'} V_f$ and $\hat{C} = \bigcup_{f\in\Gamma'} C_f$ are of fixed size (independent of I), let $n_{\hat{V}} = |\hat{V}|$ and $n_{\hat{C}} = |\hat{C}|$. We then fix bijections $var : \hat{V} \to \{1, \ldots, n_{\hat{V}}\}$ and $con : \hat{C} \to \{1, \ldots, n_{\hat{C}}\}$ such that for each $f \in \Gamma'$, there are intervals $\mathcal{V}_f = [lv_f, rv_f]$ and $\mathcal{C}_f = [lc_f, rc_f]$ such that $var(V_f) = \mathcal{V}_f$ and $con(C_f) = \mathcal{C}_f$. We assume that $\mathrm{dom}(\mathbf{I})$ is larger than $\max(n_{\hat{V}}, n_{\hat{C}})$ so that we can use number terms to index the elements of $\hat{V}$ and $\hat{C}$. There are only finitely many instances I smaller than this, and they can be handled in the interpretation Θ individually.

In defining the formulas below, for an integer interval I we write $\mu \in I$ as shorthand for the formula $\bigvee_{m\in I} \mu = m$.

The universe of $\mathbf{J}$ is a three-sorted set $\mathrm{dom}(\mathbf{J}) = U \dot\cup E \dot\cup B'$ consisting of variables U, constraints E, and bit positions B'. The set U is defined by the formula

$$\delta_U(x,\mu) = \left(x \in C \wedge \bigvee_{f\in\Gamma'} (\exists \bar y \in V^{ar(f)} : R_f(\bar y, x) \wedge \mu \in \mathcal{V}_f) \right) \quad \vee \quad (\mu = 0 \wedge x \in V).$$

In other words, the elements of U are pairs (x,μ), where $x \in V \cup C$ and μ is a number and we make the following case distinction: Either $x \in C$ and there is a constraint $x = (\bar y, f, q)$ in I, and a variable $v \in V_f$ with $var(v) = \mu$; then the pair represents one of the fresh variables in $C \times \hat V$. Or, $x \in V$ and $\mu = 0$ and the pair represents an element of V.

Similarly, the constraints E are given by

$$\delta_E(x,\mu) = x \in C \wedge \bigvee_{f\in\Gamma'} (\exists \bar y \in V^{ar(f)} : R_f(\bar y, x) \wedge \mu \in \mathcal{C}_f).$$

Again, the elements of E are pairs (x,μ), with $x \in C$ and μ an element of the number domain, and we require that if there is a constraint of the form $x = (\bar y, f, q)$, then there is a constraint $c \in C_f$ with $con(c) = \mu$.

For the domain of bit positions, we just need to make sure that the set is large enough to encode all weights in J. Taking $B' = B^2$ suffices, so

$$\delta_{B'}(x_1, x_2) = x_1, x_2 \in B$$

and we take $\phi_<(\bar x, \bar y)$ to be the formula that defines the lexicographic order on pairs.

The constraints of J are encoded in the relations R_g, $g \in \Gamma$. For an m-ary function g, this is defined by a formula ϕ_{R_g} in the free variables $(x_1, \mu_1, \ldots, x_m, \mu_m, e, \nu)$ where each (x_i, μ_i) ranges over elements of U, and (e, ν) ranges over elements of E. To be precise, we define the formula by:

$$\begin{aligned}\phi_{R_g} = \bigvee_{f\in\Gamma'} \Big(& \exists \bar y \in V^{ar(f)} : R_f(\bar y, e) \wedge \nu \in \mathcal{C}_f \\ & \wedge \bigvee_{e'=(\rho,g,r)\in C_f} \left(\nu = \ con(e') \wedge \bigwedge_{i:\rho_i\in\bar v_f} (x_i = e \wedge \mu_i = var(\rho_i)) \right. \\ & \left. \wedge \bigwedge_{i:\rho_i\notin\bar v_f} (x_i = y_i \wedge \mu_i = 0) \right) \Big).\end{aligned}$$

Finally, we define the weight relations. The weight of a constraint $\bar e = (e_1, e_2)$ is assigned the product of the weight of $e_1 \in C$ and the weight of $e_2 \in \hat C$. We have

$$\phi_{W_N}(\bar e, \bar\beta) = \bigvee_{e'\in\hat C} e_2 = con(e') \wedge \mathrm{MULT}_w(W_N(e_1, \cdot), \bar\beta),$$

where w is the numerator of the weight of the constraint e'. The definition of the denominator relation is analogous. ◀

The next lemma similarly establishes that the reduction in Lemma 4 can be realised as an FPC interpretation.

▶ **Lemma 13.** *Let Γ and Γ' be valued languages of finite size over domain D such that $\Gamma' \subseteq \Gamma_\equiv$. Then $\mathrm{VCSP}(\Gamma') \leq_{\mathrm{FPC}} \mathrm{VCSP}(\Gamma)$, by an optimum-preserving reduction.*

Proof. Note that adding constants to the value of constraints does not change the optimal solution of the instance. Hence, we only need to adapt to the scaling of the constraint functions. This can be achieved by changing the weights accordingly.

Let $I = (V, C)$ be an instance of VCSP(Γ'), given as the relational structure $\mathbf{I} = (\mathrm{dom}(\mathbf{I}), (R_f)_{f\in\Gamma'}, W_N, W_D)$. We aim to construct an instance $J = (U, E)$ of VCSP(Γ) with the same optimal solution.

The set of variables of J is V. For any $f \in \Gamma'$ we fix a function $S(f) \in \Gamma$ such that $S(f) \equiv f$. Then, the formula $\phi_{R_g}(\sigma, d) = \bigvee_{f\in\Gamma';g=S(f)} R_f(\sigma, d)$ defines the constraints of J. Let $d = (\sigma, g, r)$ be any constraint in E, and $c = (\sigma, f, q)$ be the corresponding constraint in C where $g = S(f)$, and $g = a \cdot f + b$ for some $a, b \in \mathbb{Q}$. We then set the weight r of the constraint d to be $a \cdot q$. This can again be defined by a formula in FPC. ◀

Next, we show that there is a definable reduction from VCSP(Γ) to the problem defined by a core of Γ.

▶ **Lemma 14.** *Let Γ be a valued language over D, and Γ' a core of Γ. Then,* VCSP(Γ) $\leq_{\mathrm{FO}}$ VCSP(Γ')*, by an ptimum-preserving reduction.*

Proof. Since the functions in Γ' are exactly those in Γ, only restricted to some subset of D, we can interpret any instance of VCSP(Γ) directly as an instance of VCSP(Γ'). Since the optimum of both instances is the same, by Lemma 7, this constitutes a reduction. ◀

The next two Lemmas together show that VCSP(Γ) and VCSP(Γ_c) are FPC-equivalent for a core language Γ. The proof follows closely the proof from [9] that they are polynomial-time equivalent.

▶ **Lemma 15** (Lemma 2, [9])**.** *Let Γ be a core over domain D. There exists an instance I_p of* VCSP(Γ) *with variables $V = \{x_a \mid a \in D\}$ such that $h_{id}(x_a) = a$ is an optimal solution of I_p and for every optimal solution h, the following hold:*

1. *h is injective; and*
2. *for every instance I' of* VCSP(Γ) *and every optimal solution h' of I', the mapping $s_h \circ h'$ is also an optimal solution, where $s_h(a) := h(x_a)$.*

▶ **Lemma 16.** *Let Γ be a core over a domain D of finite size. Then,* VCSP(Γ_c) $\leq_{\mathrm{FPC}}$ VCSP(Γ)*, by an optimum-preserving reduction.*

Proof. Let $I_c = (V_c, C_c)$ be an instance of VCSP(Γ_c), and let $I_p = (V_p, C_p)$ be an instance of VCSP(Γ) that satisfies the conditions of Lemma 15. We construct an instance $I = (V, C)$ of VCSP(Γ) as follows. The set of variables V is

$$V := V_c \dot\cup V_p = V_c \dot\cup \{x_a \mid a \in D\}.$$

By Definition 8, each function $f \in \Gamma_c$ is associated with some function $g = \gamma(f) \in \Gamma$, such that f is obtained from g by fixing the values of some set of variables of g. Let T_f be the corresponding index set, $t_f : T_f \to D$ the corresponding partial assignment of variables of g, and s_f the injective mapping between parameter positions of f and g. Then, we add for each constraint $c' = (\sigma', f, q) \in C_c$ the constraint $c = (\sigma, g, q)$ to C, where we replace each parameter of g that is fixed to $a \in D$ by the variable x_a, or formally, $\sigma_i = x_{t_f(i)}$ if $i \in T_f$, and $\sigma_i = \sigma'_{s_f^{-1}(i)}$ otherwise. Additionally, we add each constraint of C_p to C with its weight multiplied by some sufficiently large factor M such that every optimal solution to I, when restricted to $\{x_a \mid a \in D\}$, constitutes also an optimal solution to I_p. For instance, M can be chosen as $M := \sum_{(\sigma,g,q)\in C\setminus C_p} q \cdot \max_{f\in\Gamma_c;\bar x} f(\bar x)$. Note that since the domain and the

constraint language are finite, and the functions are finite valued, the value of $\max_{f\in\Gamma_c;\bar{x}} f(\bar{x})$ exists and is a constant. Together, the set of constraints C is defined as

$$C = \{(\sigma, g, q) \mid \exists\sigma', f : g = \gamma(f), (\sigma', f, q) \in C_c,\ \forall i \in T_f : \sigma_i = t_f(i),\ \forall i \notin T_f : \sigma_i = \sigma'_{s_f^{-1}(i)}\}$$
$$\cup \{(\sigma, g, M \cdot q) \mid (\sigma, g, q) \in C_p\}.$$

To see that this construction is a reduction, consider an optimal solution h_c of I_c. This gives rise to an optimal solution h of I, where $h(x) = h_c(x)$ for $x \in V_c$, and $h(x) = h_{id}(x)$ for $x \in V_p$. In the other direction, let h be an optimal solution to I, and its restriction to V_p, $h_p := h_{|V_p}$ is an optimal solution to I_p. By Lemma 15, the operation s_{h_p} is a permutation on D, and in particular, by repeatedly applying the second part of Lemma 15, the inverse permutation $s_{h_p}^{-1}$ is an optimal solution to I_p as well. Now, again by application of the second part of Lemma 15, we obtain an optimal solution $h' := s_{h_p}^{-1} \circ h$ to I, for which $h'(x_a) = a$ for each $a \in D$. Thus, the restriction of h' to V_c is an optimal solution to I_c.

We now formulate the above construction as an FPC interpretation.

Let I_c be given as a structure $\mathbf{I}_c$ over $\tau_{\Gamma_c} = (<, (R_f)_{\Gamma_c}, W_N, W_D)$. Furthermore, let $I_p = (V_p, C_p)$ be some fixed instance of VCSP(Γ) that satisfies the conditions of Lemma 15. We construct an FPC-interpretation $\Theta = (\bar{\delta}, \varepsilon, \phi_<, (\phi_{R_f})_{f\in\Gamma}, \phi_{W_N}, \phi_{W_D})$ that defines $\mathbf{I} = \Theta(\mathbf{I}_c)$. The universe $\mathrm{dom}(\mathbf{I}_c)$ is the three-sorted set $V_c \dot\cup C_c \dot\cup B_c$. In the same way, the universe of the structure $\mathbf{I}$ is a three sorted set $V \dot\cup C \dot\cup B$. Just as in the proof of Lemma 12, to code elements of V_p and C_p, we fix bijections $var : V_p \to \{1, \ldots, |V_p|\}$ and $con : C_p \to \{1, \ldots, |C_p|\}$

The sets V and C are then defined by the formulas

$$\delta_V(x) = x \in V_c \vee x \in \{1, \ldots, |V_p|\}; \quad \text{and} \quad \delta_C(x) = x \in C_c \vee x \in \{1, \ldots, |C_p|\}.$$

The set B needs to be large enough to code all weights. We can take $B = B_c^2$.

$$\delta_B(x_1, x_2) = x_1, x_2 \in B_c,$$

and let $\phi_<$ define the lexicographic order on B_C^2.

For each m-ary function $g \in \Gamma$, we have the formula

$$\phi_{R_g}(\bar{x}, c) = \bigvee_{e=(\rho,g,r)\in C_p} \left(c = con(e) \wedge \bigwedge_{1\le i\le m} x_i = var(\rho_i) \right)$$
$$\bigvee_{f:\gamma(f)=g} \left(\exists \bar{y} \in V_c^{\mathrm{ar}(f)} : R_f(\bar{y}, c) \wedge \bigwedge_{i\in T_f} x_i = var(t_f(i)) \bigwedge_{i\notin T_f} x_i = y_{s_f^{-1}(i)} \right).$$

The weights are given by

$$\phi_{W_N}(c, \bar{\beta}) = (c \in C_c \wedge W_N(c, \beta)) \vee \bigvee_{e=(\rho,g,r)\in C_p} (c = con(e) \wedge \mathrm{MULT}_{r\cdot L}(B_c, \bar{\beta})),$$

where L is given by

$$L = \max_{f\in\Gamma_c;\bar{x}\in D^{ar(f)}} f(\bar{x}).$$

The denominator is given by

$$\phi_{W_D}(c, \bar{\beta}) = (c \in C_c \wedge W_D(c, \beta)) \vee \bigvee_{e\in C_p} (c = con(e) \wedge \mathrm{BIT}(1, \beta)).$$

Here, another case distinction is in place. Either we have $c \in C_c$, and the weight is simply the same as given by W_N and W_D. Or, the constraint c corresponds to some constraint $e = (\rho, g, r) \in C_p$, and we assign the weight $L \cdot 2^{|B_c|} \cdot r$ to c. ◀

4 Expressibility Result

The fact that VCSP(Γ) is definable in FPC whenever Γ_c does not have the (XOR) property is obtained quite directly from Theorems 10 and 11. Here we state the result in somewhat more general form.

▶ **Theorem 17.** *For any valued constraint language* Γ *over a finite domain* D*, there is an* FPC *interpretation* Θ *of* $\tau_{\mathbb{Q}}$ *in* τ_Γ *that takes an instance* I *to a representation of the optimal value of* BLP(I)*.*

Proof. We show that it is possible to interpret BLP(I) as a τ_{LP}-structure in I by means of an FPC-interpretation. The statement then follows by Theorem 11 and the composition of FPC-reductions.

Let $I = (V, C)$ be given as the τ_Γ structure $\mathbf{I}$ with universe $\mathrm{dom}(\mathbf{I}) = V \dot\cup C \dot\cup B$. Our goal is to define a τ_{LP}-structure $\mathbf{P}$ representing BLP(I) given by $(A, \bar b, \bar c)$. The set of variables of $\mathbf{P}$ is the union of the two sets $\lambda = \{\lambda_{c,\nu} \mid c = (\sigma, f, q) \in C, \nu \in D^{|\sigma|}\}$ and $\mu = \{\mu_{x,a} \mid x \in V, a \in D\}$. In order to refer to elements of D in our interpretation, we fix a bijection $dom : D \to \{1, \dots, |D|\}$ between D and an initial segment of the natural numbers.

Then, the sets λ and μ are defined by

$$\lambda(c, \bar\nu) = \bigvee_{f \in \Gamma} \left(\exists \bar y \in V^{\mathrm{ar}(f)} : R_f(\bar y, c) \wedge \bigwedge_{1 \le i \le ar(f)} \bigvee_{a \in D} \nu_i = dom(a) \right).$$

Here, we assume that $\bar\nu$ is a tuple of number variables of length $\max_{f\in\Gamma} \mathrm{ar}(f)$. This creates some redundant variables, related to constraints whose arity is less than the maximum. We also have

$$\mu(x, \alpha) = x \in V \wedge \bigvee_{a \in D} y = dom(a).$$

For the set of linear constraints, we observe that the constraints resulting from the equalities of the form (2) can be indexed by the set

$$J_{(2)} = \{j_{c,i,a,b} \mid c = (\sigma, f, q) \in C, i \in \{1, \dots, |\sigma|\}, a \in D, b \in \{0, 1\}\},$$

since we have for each $c \in C$, $i \in \{1, \dots, |\sigma|\}$, and $a \in D$ a single equality, and hence two inequalities, one for each value of b. This can be expressed by

$$\begin{aligned} J_{(2)}(c, \iota, \alpha, \beta) = & c \in C \wedge \bigvee_{f \in \Gamma} \exists \bar y \in V^{ar(f)} : R_f(\bar y, c) \\ & \wedge \iota \le ar(f) \quad \wedge \bigvee_{a \in D} \alpha = dom(a) \quad \wedge \beta \in \{0, 1\}. \end{aligned}$$

Similarly, the constraints resulting from (3) can be indexed by $J_{(3)} = \{j_{x,b} \mid x \in V, b \in \{0, 1\}\}$, defined by the formula,

$$J_{(3)}(x, \beta) = x \in V \wedge \beta \in \{0, 1\}.$$

Finally, we have two inequalities bounding the range of each variable, indexed by $J_{(4)} = \{j_{v,b} \mid v \in \lambda \cup \mu, b \in \{0, 1\}\}$, defined by

$$J_{(4)}(\bar v, \beta) = \lambda(\bar v) \vee \mu(\bar v) \wedge \beta \in \{0, 1\}.$$

The universe $\mathrm{dom}(\mathbf{L})$ is then the three-sorted set $Q \dot\cup R \dot\cup B'$ with index sets Q and R for columns and rows respectively, and a domain for bit positions B', defined by

$$\delta_Q(\bar{x}) = \lambda(\bar{x}) \vee \mu(\bar{x}); \quad \delta_R(\bar{x}) = J_{(2)}(\bar{x}) \vee J_{(3)}(\bar{x}) \vee J_{(4)}(\bar{x}); \quad \text{and} \quad \delta_{B'}(x) = x \in B.$$

The entries in the matrix $A \in \mathbb{Q}^{Q\times R}$, and the two vectors $\bar{b} \in \mathbb{Q}^Q$ and $\bar{c} \in \mathbb{Q}^R$ consist only of elements of $\{0, 1, -1\}$ and the weight of some constraint in C. It is easily seen that these can be suitably defined in FPC. ◀

Combining this with Theorem 10 gives the positive half of the definability dichotomy.

▶ **Corollary 18.** *If Γ is a valued constraint language such that property* (XOR) *does not hold for Γ_c, then* VCSP(Γ) *is definable in* FPC.

5 Inexpressibility Result

We now turn to the other direction and show that if VCSP(Γ) is such that Γ_c has the (XOR) property then VCSP(Γ) is not definable in FPC. In fact, we will prove the stronger inexpressibility result that those VCSPs are not even definable in the stronger logic C^ω.

Our proof proceeds as follows. The main result in [12] characterizes the intractable constraint languages Γ as exactly those languages whose extension Γ_c has the property (XOR), by constructing a polynomial time reduction from MAXCUT to VCSP(Γ). We show that this reduction can also be carried out in FPC. It is then left to show that MAXCUT itself is not definable in C^ω. To this end, we describe a series of FPC-reductions from 3-SAT to MAXCUT which roughly follow known polynomial time counterparts. Finally, results of [4] and [2] establish that 3-SAT is not definable in C^ω, concluding the proof.

We consider the problem MAXCUT, where one is given an undirected graph $G = (V, E)$ along with a weight function $w : E \to \mathbb{Q}^+$ and is looking for a bipartition of vertices $p : V \to \{0, 1\}$ that maximises the payout function $b(p) = \sum_{(u,v)\in E; p(u)\neq p(v)} w(u, v)$. In the decision version of the problem, an additional constant $t \in \mathbb{Q}^+$ is given and the question is then whether there is a partition p with $b(p) \geq t$.

An instance of (decision) MAXCUT is given as a relational structure $\mathbf{I}$ over the vocabulary $\tau_{\mathrm{MAXCUT}} = (E, <, W_N, W_D, T_N, T_D)$. The universe $\mathrm{dom}(\mathbf{I})$ is a two-sorted set $U = V \dot\cup B$, consisting of vertices V, and a set B of bit positions, linearly ordered by $<$. In addition to the edge relation $E \subseteq V \times V$, there are two weight relations $W_N, W_D \subseteq V \times V \times B$ which encode the numerator and denominator of the weight between two vertices. Finally, the unary relations $T_N, T_D \subseteq B$ encode the numerator and denominator of the threshold.

▶ **Lemma 19.** *If Γ is a language for which* (XOR) *holds, then,* MAXCUT $\leq_{\mathrm{FPC}}$ VCSP($\langle\Gamma\rangle_\equiv$).

Proof. Let $I = (V, E, w, t)$ be a given MAXCUT instance. We define an equivalent instance $J = (U, C, t')$ of VCSP($\Gamma_\equiv$) as follows. Since (XOR) holds for Γ, there are two distinct elements $a, b \in D$ for which $\langle\Gamma\rangle_\equiv$ contains a binary function f, such that $f(a, b) = 1$ if $a = b$ and $f(a, b) = 0$ otherwise. By creating a variable for each vertex in V and adding a constraint $((u, v), f, w(e))$ for each edge $e = (u, v) \in E$, we obtain a VCSP with the same optimal solution. The threshold constant t' is then set to $t' = M - t$, where $M := \sum_{e\in E} w(e)$.

We now turn this into an FPC-interpretation Θ of $\tau_{\langle\Gamma\rangle_\equiv}$ in τ_{MAXCUT}. Let $\mathbf{I}$ be the relational representation of I over τ_{MAXCUT} with the two-sorted universe $V \dot\cup B$.

The structure $\mathbf{J} = \Theta(\mathbf{I})$ has a three-sorted universe $\mathrm{dom}(\mathbf{J}) = U \dot\cup C \dot\cup B'$ consisting of variables $U = V$, constraints $C = V^2$, and bit positions $B' = B \times \{1, \ldots, |E|\}$.

$$\delta_U(x) = x \in V; \quad \delta_C(x_1, x_2) = x_1, x_2 \in V; \quad \text{and} \quad \delta_{B'}(x, \mu) = x \in B \wedge \mu \leq \#_{y,z} E(y, z).$$

Since $M \leq |E| \max_{e \in E} w(e)$, and each $w(e)$ can be represented by $|B|$ bits, $|E| \cdot |B|$ bits suffice to represent the threshold $M - t$.

Each edge $e = (u, v)$ gives rise to a constraint $((u, v), e, w(e))$, which is encoded in R_f.

$$\phi_{R_f}(\bar{x}, \bar{c}) = E(\bar{x}) \wedge \bar{x} = \bar{c}.$$

The weights are simply carried over.

$$\phi_{W_N}(\bar{c}, b) = W_N(\bar{c}, b) \quad \text{and} \quad \phi_{W_D}(\bar{c}, b) = W_D(\bar{c}, b)$$

The threshold is set to $M - t$. As FPC can define any polynomial-time computable function on an ordered domain, it is possible to write formulas ϕ_{T_N} and ϕ_{T_D} defining the numerator and denominator of the threshold $M - t$ on the ordered sort B'.

The remaining relations R_g corresponding to functions in $g \in \langle \Gamma \rangle_{\equiv} \backslash \{f\}$ are empty. ◀

The next ingredient is to show that the classical series of polynomial time reductions from 3-SAT to MAXCUT can also be carried out within FPC. The chain of reductions involves three steps. The first one is a reduction from 3-SAT to 4-NAESAT (Not All Equal SAT), followed by a reduction from 4-NAESAT to 3-NAESAT, and finally 3-NAESAT is reduced to MAXCUT. We begin with defining the relational representations of these problems.

An instance of 3-SAT is given as a relational structure over the vocabulary $\tau_{\text{3SAT}} = (R_{000}, \ldots, R_{111})$ with eight ternary relations. The universe of the instance is a set of variables V and we say the instance is satisfiable if there is a map $h : V \to \{0, 1\}$ such that for any $a, b, c \in V$ if $(a, b, c) \in R_{ijk}$ then it is *not the case* that $h(a) = i$ and $h(b) = j$ and $h(c) = k$. Similarly, 3-NAESAT is a class of structures over $\tau_{\text{3NAESAT}} = (N_{000}, \ldots, N_{111})$ and an instance over the universe V is satisfiable if there is a map $h : V \to \{0, 1\}$ such that for any $a, b, c \in V$ if $(a, b, c) \in N_{ijk}$ then the values of $h(a) \oplus i$, $h(b) \oplus j$ and $h(c) \oplus k$ are not all the same. Finally, 4-NAESAT is defined similarly over the vocabulary $\tau_{\text{4NAESAT}} = (N_{0000}, \ldots, N_{1111})$ with sixteen 4-ary relations.

▶ **Lemma 20.** 3-SAT $\leq_{\text{FPC}}$ MAXCUT.

Proof. 3-SAT $\leq_{\text{FO}}$ 4-NAESAT: Let $\mathbf{I} = (V, R_{000}, \ldots, R_{111})$ be any given 3-SAT instance. Consider a 4-NAESAT instance $\mathbf{J} = (U, N_{0000}, \ldots, N_{1111})$ with $V \subset U$, i.e. there is at least one variable in U not contained in V. Furthermore, let $(a, b, c, z) \in N_{ijk0}$ hold if, and only if, $(a, b, c) \in R_{ijk}$ and $z \in U \backslash V$, and let the relations N_{ijk1} be empty. The instance $\mathbf{J}$ is now satisfiable if, and only if, $\mathbf{I}$ is satisfiable: Whenever there is a satisfying assignment for $\mathbf{I}$, the same assignment extended with $z = 0$ for all $z \in U \backslash V$ will also be a satisfying assignment for $\mathbf{J}$. In the other direction, if there is a satisfying assignment for $\mathbf{J}$, there is always a satisfying one that sets $z = 0$ for all $z \in U \backslash V$, since negating every variable does not change the value of a NAE-clause, and each clause only contains one variable in $U \backslash V$. In terms of a FPC-interpretation, this construction looks as follows.

We take as universe $\text{dom}(\mathbf{J})$ the set V^2, and interpret an element (a, a) as representing the variable $a \in V$, and any element $(a, b), a \neq b$ as a fresh variable in $U \backslash V$.

$$\delta_U(x_1, x_2) = x_1, x_2 \in V$$

$$\phi_{N_{ijk0}}(\bar{x}, \bar{y}, \bar{z}, \bar{w}) = R_{ijk}(x_1, y_1, z_1) \wedge w_1 \neq w_2 \wedge \bigwedge_{\bar{v} \in \{\bar{x}, \bar{y}, \bar{z}\}} v_1 = v_2$$

$$\phi_{N_{ijk1}}(\bar{x}, \bar{y}, \bar{z}, \bar{w}) = \text{False}$$

4-NAESAT $\leq_{\text{FPC}}$ 3-NAESAT: Let $\mathbf{I} = (V, N_{0000}, \ldots, N_{1111})$ be an instance of 4-NAESAT. Note that we can split every clause $\text{NAE}(a, b, c, d)$ into two smaller 3-NAESAT clauses $\text{NAE}(a, b, z)$ and $\text{NAE}(\neg z, c, d)$ for some fresh variable z. The following interpretation realises this conversion.

In order to introduce a fresh variable for each clause of the 4-NAESAT instance, the universe of the 3-NAESAT instance will consist of tuples from $V^4 \times \{0, 1\}^5$, where the first eight components encode a clause in $\mathbf{I}$ and the last component is a flag indicating whether the element represents a fresh variable or one that appears already in V. The convention is then that an element of the form $(a, a, a, a, 0, \ldots, 0)$ represents the variable $a \in V$, and an element of the form $(a, b, c, d, i, j, m, n, 1)$ represents the fresh variable that is used to split the clause $N_{ijmn}(a, b, c, d)$. The remaining relations are defined as empty.

3-NAESAT $\leq_{\text{FPC}}$ MAXCUT: The following construction transforms a given 3-NAESAT instance $\mathbf{I} = (V, N_{000}, \ldots, N_{111})$ into an equivalent (decision) MAXCUT instance $\mathbf{J} = (\text{dom}(\mathbf{J}), E, W_N, W_D, T_N, T_D)$. Let m be the number of clauses in $\mathbf{I}$, and fix $M := 10m$. For each variable $v \in V$, we have two vertices denoted v_0 and v_1, in our graph, along with an edge (v_0, v_1) of weight M. For each tuple $(x, y, z) \in N_{ijk}$ we add a triangle between the vertices x_i, y_j, and z_k with edge-weight 1. Setting the cut threshold to $t := |V| \cdot M + 2m$ gives us an equivalent instance: If $\mathbf{I}$ is satisfiable, say by an assignment f, then the partition given by $p(v_i) = f(v) + i \bmod 2$ cuts through every edge of the form (v_0, v_1), and through two edges in every triangle, resulting in a payout of $|V| \cdot M + 2m$. On the other hand, any bipartition of payout larger or equal to $|V| \cdot M + 2m$ has to cut through all edges of the form (v_0, v_1), since it can only cut through two edges in each triangle. Hence, any such bipartition induces a satisfying assignment to the 3-NAESAT instance. We use the following FPC-interpretation to realise this construction.

The universe of $\mathbf{J}$ is defined as a two-sorted set $\text{dom}(\mathbf{J}) = U \dot\cup B$, consisting of vertices $U = V \times \{0, 1\}$ and bit positions $B = \{1, \ldots, \alpha\}$ for some sufficiently large α. In particular, α has to be chosen larger than $\log_2 t$. Since m is at most $|V|^3$, taking $\alpha = |V|^4$ suffices.

$$\delta_U(x_1, x_2) = x_1 \in V, x_2 \in \{0, 1\} \quad \text{and} \quad \delta_B(\bar\mu) = \bigwedge_{1 \leq i \leq 4} \mu_i \leq \#_v v \in V.$$

The edge relation is given by

$$\phi_E(\bar x, \bar y) = x_1 = y_1 \wedge x_2 \neq y_2 \bigvee_{i,j,k \in \{0,1\}} \exists u, v, w \in V : N_{ijk}(u, v, w) \wedge \bar x, \bar y \in \{(u, i), (v, j), (w, k)\}.$$

The edge weights and the cut threshold are defined by

$$\phi_{W_N}(\bar x, \bar y, \beta) = x_1 = y_1 \wedge x_2 \neq y_2 \wedge \text{BIT}(1, \beta) \vee \text{BIT}\left(10 \cdot \sum_{i,j,k \in \{0,1\}} \#_{u,v,w} N_{ijk}(u, v, w), \beta\right),$$

$$\phi_{T_N}(\beta) = \text{BIT}\left((2 + 10 \cdot \#_v v \in V) \cdot \sum_{i,j,k \in \{0,1\}} \#_{u,v,w} N_{ijk}(u, v, w), \beta\right),$$

$$\phi_{W_D}(\bar x, \bar y, \beta) = \text{BIT}(1, \beta),$$

$$\phi_{T_D}(\beta) = \text{BIT}(1, \beta).$$

Note that the weights and the cut threshold are integer, hence the denominator relations simply code 1. ◀

▶ **Lemma 21.** 3-SAT *is not expressible in* C^ω.

Proof. Note that a 3-SAT instance $\mathbf{I} = (V, R^{\mathbf{I}}_{000}, \dots, R^{\mathbf{I}}_{111})$ can also be interpreted as an instance of CSP($\Gamma_{3\mathrm{SAT}}$) for $\Gamma_{3SAT} = \{R_{000}, \dots, R_{111}\}$ and $R_{ijk} = \{0,1\}^3 \backslash (i,j,k)$. Hence, we can apply results from the algebraic classification of CSPs to determine the definability of 3-SAT. In this context, it has been shown in [4] that the algebra of polymorphisms corresponding to Γ_{3SAT} contains only essentially unary operations. It follows from the result in [2] that 3-SAT is not definable in C^ω. ◀

▶ **Theorem 22.** *Let* Γ *be a valued constraint language of finite size and let* Γ' *be a core of* Γ. *If* (XOR) *holds for* Γ'_c, *then* VCSP(Γ) *is not expressible in* C^ω.

Proof. Assume property (XOR) holds for Γ'_c. By Lemma 19, MAXCUT FPC-reduces to VCSP($\langle\Gamma'_c\rangle_\equiv$). Lemmas 12 to 16 provide a chain of FPC-reductions from VCSP($\langle\Gamma'_c\rangle_\equiv$) to VCSP($\Gamma$). Since C^ω is closed under FPC-reductions, Lemmas 20 and 21 together show that MAXCUT is not definable in C^ω, and hence neither is VCSP(Γ). ◀

6 Constraint Languages of Infinite Size

In representing the problem VCSP(Γ) as a class of relational structures, we have chosen to fix a finite relational signature τ_Γ for each finite Γ. An alternative, *uniform* representaation would be to fix a single signature which allows for the representation of instances of VCSP(Γ) for arbitrary Γ by coding the functions in Γ explicitly in the instance. In this section, we give a description of how this can be done. Our goal is to show that our results generalise to this case, and that the definability dichotomy still holds.

Let Γ now be a valued constraint language over some finite domain D. The challenge of fixing a relational signature for instances of VCSP(Γ) is that different instances may use different sets of functions of Γ in their constraints, and hence, we cannot represent each function as a relation in the signature. Instead, we make the functions part of the universe, together with tuples over D of different arities as their input. Let I be an instance of VCSP(Γ) where the constraints use functions from a finite subset $\Gamma_I \subset \Gamma$, and let m be the maximal arity of any function in Γ_I. We then represent I as a structure $\mathbf{I}$ with the multi-sorted universe $\mathrm{dom}(\mathbf{I}) = V \dot\cup C \dot\cup B \dot\cup F \dot\cup T$, where V is a set of variables, C a set of constraints, B a set of numbers on which we have a linear order, F a set of function symbols corresponding to functions in Γ_I, and T is a set of tuples from $D \cup D^2 \cup \ldots \cup D^m$, over the signature $\tau_D = (<, R_{\mathrm{fun}}, R_{\mathrm{scope}}, W_N, W_D, Def_N, Def_D, Enc)$. Here, the relations encode the following information.

- $R_{\mathrm{fun}} \subseteq C \times F$: This relation matches functions and constraints, i.e. $(c,f) \in R_{\mathrm{fun}}$ denotes that $c = (\sigma, f, q)$ is a constraint of the instance for some scope σ and weight q.
- $R_{\mathrm{scope}} \subseteq C \times V \times B$: This relation fixes the scope of a constraint, i.e. $(c, v, \beta) \in R_{\mathrm{scope}}$ denotes that $c = (\sigma, f, q)$ is a constraint for some function f and weight q, where the β-th component of σ is v.
- $W_N, W_D \subseteq C \times B$: This is analogous to the finite case. These two relations together encode the rational weights of the constraints.
- $Def_N, Def_D \subseteq F \times T \times B$: These two relations together fix the definition of some function symbol in F. That is, $(f, t, \beta) \in Def_N$ denotes that the β-th bit of the numerator of the value of f on input t is 1, and similarly for Def_D and the denominator.
- $Enc \subseteq T \times D \times B$: This relation fixes the encoding of tuples as elements in T, i.e. $(t, a, \beta) \in Enc$ denotes that the β-th component of the tuple t is the element $a \in D$.

The above signature allows now for instances I, I' with different sets of functions Γ_I and $\Gamma_{I'}$ to be represented as structures of the same vocabulary. Since the set of function symbols is part of the universe, the relations Def_N, Def_D are required to give concrete meaning to these function symbols.

We now say, for a (potentially infinite) valued constraint language Γ that VCSP(Γ) is *uniformly definable* in FPC if there is an FPC-interpretation of $\tau_{\mathbb{Q}}$ in τ_D which takes an instance $\mathbf{I}$ of VCSP(Γ) to the cost of its optimal solution. Our inexpressibility result, Theorem 22, immediately carries over to this setting as it is easy to construct an FPC reduction from the τ_Γ representation of VCSP(Γ) to the τ_D representation.

▶ **Theorem 23.** *Let Γ be a valued constraint language and let Γ' be a core of Γ. If* (XOR) *holds for Γ'_c, then* VCSP(Γ) *is not uniformly definable in C^ω.*

For the positive direction, i.e. to show that VCSP(Γ) is *uniformly definable* in FPC in all other cases, we simply need to adapt the proof of Theorem 17 to fit the new representation.

▶ **Theorem 24.** *Let Γ be a valued constraint language and let Γ' be a core of Γ. If* (XOR) *does not hold for Γ'_c, then* VCSP(Γ) *is uniformly definable in* FPC.

References

1. M. Anderson, A. Dawar, and B. Holm. Maximum matching and linear programming in fixed-point logic with counting. In *Proceedings of the 28th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 173–182, 2013.
2. A. Atserias, A. Bulatov, and A. Dawar. Affine systems of equations and counting infinitary logic. *Theoretical Computer Science*, 410(18):1666–1683, 2009.
3. L. Barto and M. Kozik. Constraint satisfaction problems solvable by local consistency methods. *J. ACM*, 61, 2014.
4. A. Bulatov, P. Jeavons, and A. Krokhin. Classifying the complexity of constraints using finite algebras. *SIAM Journal on Computing*, 34(3):720–742, 2005.
5. D. Cohen, M.C. Cooper, P. Jeavons, and A. Krokhin. The complexity of soft constraint satisfaction. *Artificial Intelligence*, 170(11):983–1016, 2006.
6. A. Dawar. The nature and power of fixed-point logic with counting. *ACM SIGLOG News*, 2:8–21, 2015.
7. H-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer, 2nd edition, 1999.
8. T. Feder and M.Y. Vardi. Computational structure of monotone monadic SNP and constraint satisfaction: A study through Datalog and group theory. *SIAM Journal on Computing*, 28:57–104, 1998.
9. A. Huber, A. Krokhin, and R. Powell. Skew bisubmodularity and valued CSPs. *SIAM Journal on Computing*, 43(3):1064–1084, 2014.
10. L. Libkin. *Elements of Finite Model Theory*. Springer, 2004.
11. M. Otto. *Bounded Variable Logics and Counting – A Study in Finite Models*, volume 9 of *Lecture Notes in Logic*. Springer-Verlag, 1997.
12. J. Thapper and S. Živný. The complexity of finite-valued CSPs. In *Proceedings of the 45th ACM Symposium on the Theory of Computing*, STOC'13, pages 695–704. ACM, 2013.

Evidence for Fixpoint Logic

Sjoerd Cranen, Bas Luttik, and Tim A. C. Willemse

Technische Universiteit Eindhoven
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
{s.cranen,s.p.luttik,t.a.c.willemse}@tue.nl

Abstract

For many modal logics, dedicated model checkers offer diagnostics (*e.g.*, counterexamples) that help the user understand the result provided by the solver. Fixpoint logic offers a unifying framework in which such problems can be expressed and solved, but a drawback of this framework is that it lacks comprehensive diagnostics generation. We extend the framework with a notion of evidence, which can be specialised to obtain diagnostics for various model checking problems, behavioural equivalence and refinement checking problems. We demonstrate this by showing how our notion of evidence can be used to obtain diagnostics for the problem of deciding stuttering bisimilarity. Moreover, we show that our notion generalises the existing notions of counterexample and witness for LTL and ACTL* model checking.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases fixpoint logic, diagnostics, counterexample, model checking, stuttering bisimilarity, ACTL*

Digital Object Identifier 10.4230/LIPIcs.CSL.2015.78

1 Introduction

Many of the techniques and tools developed for the purpose of verification of software and hardware systems – such as model checkers for various temporal logics, and refinement checkers for various behavioural equivalences and preorders – perform checks that can also be encoded in fixpoint extensions of first-order logic. Rather than having specialised tools to compute answers to specific verification problems, one can then use a solver for fixpoint logic to compute answers to such problems. This approach is, *e.g.*, taken by the mCRL2 tool set [7], which solves model checking problems for transition systems (which are essentially described by first-order structures) by translating the model checking problem to the problem of checking validity of a formula in fixpoint logic [15]; in a similar vein, the tool set offers tools that decide whether there is a behavioural equivalence between two transition systems by translating the decision problem to fixpoint logic [3].

While using fixpoint logic as a unifying framework for verification problems is desirable from a theoretical point of view, there are some practical aspects that need to be considered to prepare such a framework for large scale use. One pertinent problem is the issue of diagnostic generation. Many specialised tools for, *e.g.*, LTL model checking, provide diagnostics (for instance in the form of counterexample traces). For fixpoint logic in general, no such notion yet exists to our knowledge, and it is not immediately obvious how diagnostics generation for specific verification problems can be fit into the more generic framework of fixpoint logic.

Our contribution is an approach to diagnostic generation in fixpoint logic. As a starting point for our investigation, we use the notion of a proof graph [9] for a particular fixpoint logic called *parameterised Boolean equation systems* (PBES) [16]. Proof graphs for PBESs are loosely based on the notion of *support set* of Tan and Cleaveland [23], and provide a

24th EACSL Annual Conference on Computer Science Logic (CSL 2015).
Editor: Stephan Kreutzer; pp. 78–93

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

formal representation of an argument for the validity or invalidity of a fixpoint formula in a first-order structure. They capture only relevant information about predicates defined by fixpoints, and their interdependencies; in particular, they abstract from general first-order reasoning. To build a general theory of evidence for fixpoint logics based on proof graphs, we need to adapt the notion of proof graph proposed in [9] in two ways:

- we include nodes to reveal information about first-order predicates in our proof graphs, capturing some extra information needed for evidence extraction, and
- we integrate the concept of negation (of fixpoint formulas) into our proof graphs, which was not yet needed in the context of PBES.

Intuitively, evidence is that part of a first-order structure that captures all relevant information in an argument for the validity or invalidity of a fixpoint formula. Given a proof graph representing such an argument, we propose to define evidence as a weak substructure in which all information referred to in the proof graph is still available. The weakest such substructure we call the evidence projection associated with the proof graph.

We confirm the usefulness of our definition of evidence by suitably instantiating it to get formal notions of counterexample and witness in the context of behavioural equivalence checking, and in the context of model checking. First, we define stuttering bisimilarity on Kripke structures as a general fixpoint formula, and then show how a counterexample can be obtained using the notion of evidence presented earlier. Second, we show how to retrieve counterexamples of a linear form (traces) for LTL model checking, and tree-like ones (as presented by Clarke et al. in [4]) for ∀CTL*/∃CTL* model checking.

The theory of proof graphs has much in common with other formal systems for representing proofs in the context of logics and calculi including notions of fixpoints or inductively defined predicates, such as the various tableaux systems for fixpoint logics [17, 11], proof systems for recursive types (see, *e.g.*, [12] for an overview), and the proof systems for inductively defined predicates discussed in [1]. It also has a close resemblance with parity games [13]. The notion of proof graph, however, serves a different purpose, as a stepping stone towards a formal notion of evidence for fixpoint logics with a focus on the predicates defined as fixpoints. It therefore only captures relevant information about first-order predicates, predicates defined by fixpoints, and their interdependencies; in particular, it abstracts from general first-order reasoning.

To simplify the presentation, we discuss our results in the context of *Least Fixpoint Logic* (LFP) which we introduce in Section 2.[1] In Section 3 we recap and adapt the notion of proof graph, and extend it with a notion of evidence for LFP. In Section 4.1, we first define stuttering bisimilarity on Kripke structures as a general LFP formula, and then show how a counterexample can be obtained using our notion of evidence. We illustrate how our proof graphs can also be used to generate counterexamples and witnesses for model checking problems in Section 4.2, and we conclude in Section 5.

2 Least fixpoint logic

We recall the syntax and semantics of LFP, an extension of first-order logic with least and greatest fixpoint operators, and we fix some additional notation and concepts needed in the rest of the paper.

[1] In [6] it is shown that they also hold for a more general fixpoint logic that encompasses both LFP and PBES.

Let $\Sigma = \langle \mathcal{R}, \mathcal{F}, \mathsf{ar} \rangle$ be a *signature* consisting of a set of relation symbols $\mathcal{R}$, a set of function symbols $\mathcal{F}$, and a function $\mathsf{ar} \colon \mathcal{R} \cup \mathcal{F} \to \mathbb{N}$ that assigns an *arity* to every symbol in $\mathcal{R}$ and $\mathcal{F}$. Furthermore, we assume a countably infinite set of *first-order variables*. We inductively define a *term* to be either a first-order variable, or a function symbol of arity n applied to a sequence of n terms; a term is *open* if it contains variables, and *closed* otherwise. We also presuppose a set $\mathcal{X}$ of *second-order variables*, each with an associated arity; a second-order variable X of arity n can be thought of as ranging over n-ary relations. Then, the set of *LFP formulas* over Σ is defined by the following grammar:

$$\varphi, \psi ::= X\bar{t} \mid R\bar{t} \mid t_1 = t_2 \mid \neg\varphi \mid \varphi \vee \psi \mid \exists_x \varphi \mid [\mathbf{lfp}\, X\bar{x}.\, \varphi]\bar{t}$$

with X ranging over the second-order variables in $\mathcal{X}$, R ranging over the relation symbols in $\mathcal{R}$, x ranging over first-order variables, t_1 and t_2 ranging over terms, and $\bar{x}$ and $\bar{t}$ ranging over finite sequences of first-order variables and terms of appropriate length. Not all LFP formulas generated by the grammar above are considered *well-formed*; for formulas of the form $[\mathbf{lfp}\, X\bar{x}.\, \varphi]\bar{t}$ we impose the additional syntactic requirement that all occurrences of X in φ are *positive*, *i.e.*, in the scope of an even number of negations. Henceforth, we will only consider well-formed LFP formulas. We write $\varphi \sqsubseteq \psi$ if φ is a subformula of ψ.

A *(first-order) structure* $\mathfrak{A}$ is a tuple $\langle A, \Sigma, \mathcal{I} \rangle$ in which A is a set (the *domain of discourse*), Σ is a signature and $\mathcal{I}$ is an *interpretation function*. The interpretation function $\mathcal{I}$ is a mapping that associates with every relation symbol R in Σ a relation $R^{\mathfrak{A}}$ on A of appropriate arity, and with every function symbol f in Σ a function $f^{\mathfrak{A}}$ on A of appropriate arity.

A structure $\mathfrak{B}$ is a *weak substructure* of structure $\mathfrak{A}$, denoted $\mathfrak{B} \sqsubseteq_{\mathrm{w}} \mathfrak{A}$, if it has a domain of discourse $B \subseteq A$, the same relation and function symbols as $\mathfrak{A}$, and an interpretation function such that for every n-ary relation symbol R we have $R^{\mathfrak{B}} \subseteq R^{\mathfrak{A}} \cap B^n$, and for every n-ary function symbol f we have that $f^{\mathfrak{B}}(\bar{b}) = f^{\mathfrak{A}}(\bar{b}) \in B$ for all $\bar{b} \in B^n$.

Terms and LFP formulas are evaluated on a first-order structure $\mathfrak{A}$ in a given *environment* θ that maps first-order variables to elements of A and second-order variables to relations on A of appropriate arity. If t is a term, then $t^{\mathfrak{A},\theta}$ is the element of A denoted by t in environment θ defined in the usual way (*i.e.*, if t is a variable, then $t^{\mathfrak{A},\theta} = \theta(t)$, and if $t = f(t_0, \ldots, t_n)$, then $t^{\mathfrak{A},\theta} = f^{\mathfrak{A}}(t_0^{\mathfrak{A},\theta}, \ldots, t_n^{\mathfrak{A},\theta})$). The evaluation of LFP formulas is less straightforward due to the fixpoint operators. The idea is that, given a second-order variable X of arity n, a sequence of first-order variables $\bar{x}$ of length n, and an environment θ, we can associate with every formula φ an operator $\mathbf{T}^{\mathfrak{A},\theta}_{X,\bar{x},\varphi}$ on the complete lattice of n-ary relations on A. If all occurrences of X in φ are positive, then the operator $\mathbf{T}^{\mathfrak{A},\theta}_{X,\bar{x},\varphi}$ – which is to be defined precisely below – is monotone on the lattice of n-ary relations on A. Therefore, it has a unique least fixpoint (see [24]), which we will denote by $\mathbf{lfp}\, \mathbf{T}^{\mathfrak{A},\theta}_{X,\bar{x},\varphi}$. We proceed to define the relation $\mathfrak{A}, \theta \models \varphi$ and the operators $\mathbf{T}^{\mathfrak{A},\theta}_{X,\bar{x},\varphi} \colon A^n \to A^n$ (n the arity of X) simultaneously by induction on the structure of φ:

$$\begin{array}{llll}
\mathfrak{A}, \theta \models R\bar{t} & \text{iff } \bar{t}^{\mathfrak{A},\theta} \in R^{\mathfrak{A}} & \mathfrak{A}, \theta \models X\bar{t} & \text{iff } \bar{t}^{\mathfrak{A},\theta} \in \theta(X) \\
\mathfrak{A}, \theta \models t_1 = t_2 & \text{iff } t_1^{\mathfrak{A},\theta} = t_2^{\mathfrak{A},\theta} & \mathfrak{A}, \theta \models \neg\varphi & \text{iff not } \mathfrak{A}, \theta \models \varphi \\
\mathfrak{A}, \theta \models \varphi \vee \psi & \text{iff } \mathfrak{A}, \theta \models \varphi \text{ or } \mathfrak{A}, \theta \models \psi & \mathfrak{A}, \theta \models \exists_x \varphi & \text{iff } \mathfrak{A}, \theta[x \mapsto a] \models \varphi \text{ for some } a \in A \\
\multicolumn{4}{l}{\mathfrak{A}, \theta \models [\mathbf{lfp}\, X\bar{x}.\, \varphi]\bar{t} \text{ iff } \bar{t} \in \mathbf{lfp}\, \mathbf{T}^{\mathfrak{A},\theta}_{\varphi,X,\bar{x}}, \text{ where } \mathbf{T}^{\mathfrak{A},\theta}_{\varphi,X,\bar{x}}(R) = \{\bar{a} \mid \mathfrak{A}, \theta[X \mapsto R, \bar{x} \mapsto \bar{a}] \models \varphi\}}
\end{array}$$

If $\mathfrak{A}, \theta \models \varphi$, then we say that φ is *valid* in $\mathfrak{A}$ and θ. If $\mathfrak{A}, \theta \models \varphi$ for all environments θ, then we simply say that φ is *valid* in $\mathfrak{A}$ and write $\mathfrak{A} \models \varphi$. We write $\mathfrak{A}, \theta \not\models \varphi$ to indicate that $\mathfrak{A}, \theta \models \varphi$ does not hold.

To get a more succinct presentation, whenever $\mathfrak{A}$, θ, φ, X and $\bar{x}$ are clear from the

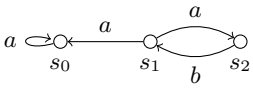


Figure 1 An LTS as a first-order structure.

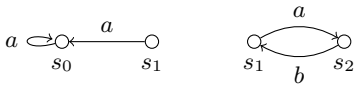


Figure 2 Evidence for φ_1 and φ_2 in $\mathfrak{A}$.

context, we will omit the super- and subscripts of $\mathbf{T}^{\mathfrak{A},\theta}_{\varphi,X,\bar{x}}$ and simply write $\mathbf{T}$. For an in-depth treatment of fixpoint theory, see *e.g.* [19, 20].

The notions of free and bound variables of a formula φ are defined as usual. By $\mathsf{fv}(\varphi)$ we denote the set of all variables with a free occurrence in φ. We sometimes need to express that environments θ and θ' agree on the free variables of some formula φ; we write $\theta \equiv_{\mathsf{fv}(\varphi)} \theta'$ if $\theta(x) = \theta'(x)$ for all first-order variables $x \in \mathsf{fv}(\varphi)$ and $\theta(X) = \theta'(X)$ for all second-order variables $X \in \mathsf{fv}(\varphi)$. Furthermore, we assume a *unique-naming convention*, by which a variable does not have both free and bound occurrences in the formula, and by which a variable is only bound by a single binder. Only in a few concrete examples we deviate from this convention in favour of readability.

The constructs $\wedge$, $\forall\_\ \_$ and $[\mathbf{gfp}\ \_\ \_.\ \_]\_$ are usually treated as syntactic abbreviations: $\varphi \wedge \psi$ for $\neg(\neg\varphi \vee \neg\psi)$, $\forall_x\ \varphi$ for $\neg\exists_x\ \neg\varphi$, and $[\mathbf{gfp}\, X\bar{x}.\, \varphi]\bar{t}$ for $\neg[\mathbf{lfp}\, X\bar{x}.\, \neg\varphi[X \mapsto \neg X]]\bar{t}$, where $\varphi[X \mapsto \neg X]$ denotes the formula obtained from φ by replacing all free occurrences of X in φ by $\neg X$. For our theory of evidence, to be presented in the next section, it will be convenient, however, to associate proof graphs directly with formulas in the extended syntax, including, in particular, the syntactic construct $[\mathbf{gfp}\ \_\ \_.\ \_]\_$ as first-class citizen.

If a formula ψ has a subformula of the form $[\sigma\, X\bar{x}.\, \chi]\bar{t}$ (with $\sigma \in \{\mathbf{lfp}, \mathbf{gfp}\}$), then we say that X is *defined* in ψ, refer to the subformula $[\sigma\, X\bar{x}.\, \chi]\bar{t}$ as the subformula *defining* X, and refer to χ as the definition of X. We also let $\sigma_{\psi,X} = \sigma$, $\bar{x}_{\psi,X} = \bar{x}$ and $\varphi_{\psi,X} = \chi$. Usually, the intended formula ψ will be clear from the context, and can therefore be safely omitted as a subscripts in these notations. By $\mathsf{dv}(\psi)$ we denote the set of all second-order variables defined in ψ. The formula ψ induces a partial order $<_\psi$ on $\mathsf{dv}(\psi)$ defined by $X <_\psi Y$ if, and only if, the subformula defining Y is a subformula of the subformula defining X.

▶ **Example 1.** A labelled transition system can be seen as a first-order structure with the set of states as domain of discourse, a binary relation symbol $\xrightarrow{a}$ for every transition label a, and a constant symbol for every state (see Figure 1).

LFP formulas can be used to express properties of a transition system. For instance,

$$\varphi_1 = [\mathbf{lfp}\, Xs.\, \forall_{s'}\ s \xrightarrow{a} s' \implies Xs']s_1$$

expresses that only finitely many consecutive a-transitions can be taken from the state denoted by the constant s_1; this formula is not valid on the labelled transition system in Figure 1. The LFP formula

$$\varphi_2 = [\mathbf{gfp}\, Xs.\, [\mathbf{lfp}\, Ys'.\, \exists_{s''}\ (s' \xrightarrow{a} s'' \wedge Ys'') \vee (s' \xrightarrow{b} s'' \wedge Xs'')]s]s_1$$

expresses – about a system in which every transition is labelled with either a or b – that from the state s_1 there is a path with infinitely many b-transitions; this formula is valid on the labelled transition system in Figure 1.

3 Evidence based on proof graphs

In this section, we adapt proof graphs, which were originally introduced in [9] for PBESs, to the setting of LFP. Intuitively, a proof graph serves to capture the essence of a reasoning

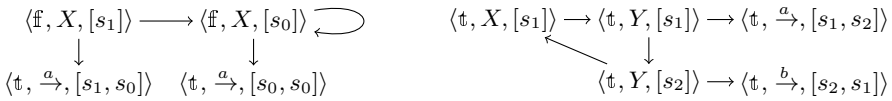


Figure 3 Proof graphs for $\mathfrak{A} \not\models \varphi_1$ and $\mathfrak{A} \models \varphi_2$.

that proves or refutes the validity of an LFP formula in a first-order structure. The notion of proof graph will be tailored for the purpose of identifying evidence for the validity or invalidity of an LFP formula inside the first-order structure. Such evidence is a, preferably smaller, weak substructure that still admits the reasoning for the validity or invalidity as captured by the proof graph.

Before we formally introduce the notion of proof graph and the associated notion of evidence, we illustrate the idea by means of some examples.

▶ **Example 2.** Consider the LFP formulas of Example 1. To support our claim that φ_1 is not valid on the labelled transition system in Figure 1, we construct the proof graph depicted on the left in Figure 3. Let us represent by $\langle \mathbb{f}, X, [s_1] \rangle$ the statement that the relation defined by X in φ_1 does not hold on s_1. According to the definition of X in φ_1, to refute that X holds on s_1 it is enough to find a state s' such that $s_1 \xrightarrow{a} s'$ and then refute that X holds on s'. The state s_0 appears to be a good candidate for s'; in our proof graph we will therefore include dependencies from $\langle \mathbb{f}, X, [s_1] \rangle$ to both $\langle \mathbb{t}, \xrightarrow{a}, [s_1, s_0] \rangle$ (expressing that $s_1 \xrightarrow{a} s_0$) and $\langle \mathbb{f}, X, [s_0] \rangle$ (expressing that X does not hold on s_0). Similarly, to refute that X holds on s_0 it suffices to include a dependency from $\langle \mathbb{f}, X, [s_0] \rangle$ to $\langle \mathbb{t}, \xrightarrow{a}, [s_0, s_0] \rangle$ and to itself.

Note that the reasoning expressed by this graph only involves the states s_0 and s_1 and the transitions $s_0 \xrightarrow{a} s_0$ and $s_1 \xrightarrow{a} s_0$, which suggests that the weak substructure of $\mathfrak{A}$ depicted on the left in Figure 2 provides sufficient evidence for the invalidity of φ_1 in $\mathfrak{A}$.

A similar reasoning can be held to construct the proof graph on the right in Figure 3 to show that φ_2 is valid in $\mathfrak{A}$, resulting in the evidence depicted on the right in Figure 2.

The proof graphs constructed in the example above explain the validity or invalidity of an LFP formula in a presupposed first-order structure in terms of primitive relations defined directly on the structure and the relations defined as fixpoints in the formula. Thus, they abstract from the standard first-order reasoning involved. Proof graphs are going to be the starting point for our theory of diagnostics for LFP formulas. We shall define evidence for the validity or invalidity of an LFP formula φ in a first-order structure $\mathfrak{A}$, based on a proof graph G, as a weak substructure of $\mathfrak{A}$ for which G is still a proof graph. For the approach to be valid, the information about $\mathfrak{A}$ that is included in G should be both correct and sufficient. Correctness means that whenever a proof graph includes a node $\langle \alpha, V, \bar{a} \rangle$, then, depending on whether α is $\mathbb{t}$ or $\mathbb{f}$, the sequence $\bar{a}$ should or should not be in the relation on $\mathfrak{A}$ denoted by V. Sufficiency means that G includes enough information from $\mathfrak{A}$ to facilitate reconstruction of the reasoning reflected by G in every weak substructure of $\mathfrak{A}$ that inherits at least that information from $\mathfrak{A}$.

We shall now first formally define the notion of proof graph for a first-order structure $\mathfrak{A}$, an LFP formula φ, and an environment θ, and then explain how it gives rise to a formal notion of evidence.

3.1 Proof graphs

The nodes of a proof graph are tuples of the form $\langle \alpha, V, \bar{a} \rangle$ in which α is either $\mathbb{t}$ or $\mathbb{f}$, V is either a relation symbol or a second-order variable, and $\bar{a}$ is a sequence of elements of $\mathfrak{A}$ the

length of which corresponds to the arity of V. We denote by $\mathcal{S}$ the set of all such nodes, and, for $\mathcal{Y} \subseteq \mathcal{R} \cup \mathcal{X}$, we denote by $\mathcal{S}_\mathcal{Y}$ the subset of $\mathcal{S}$ consisting of all nodes of which the second element is from $\mathcal{Y}$, *i.e.*,

$$\mathcal{S}_\mathcal{Y} = \{\langle \alpha, V, \bar{a}\rangle \in \{\mathbb{t}, \mathbb{f}\} \times \mathcal{Y} \times A^* \mid \mathsf{ar}(X) = |\bar{a}|\}.$$

We will often use the subscript to refer to nodes in which only specific relations or variables occur, *e.g.*, $S_{\{X\}}$ for all nodes of which the second element is the second-order variable X. To concisely express that the statement represented by a node $\langle \alpha, V, \bar{a}\rangle$ is true in a structure $\mathfrak{A}$ and an environment θ, we write

$$\mathfrak{A}, \theta \models \langle \alpha, V, \bar{a}\rangle \quad \text{for} \quad \begin{cases} \bar{a} \in \theta(V) \iff \alpha = \mathbb{t} & \text{if } V \in \mathcal{X}, \text{ and} \\ \bar{a} \in V^{\mathfrak{A}} \iff \alpha = \mathbb{t} & \text{if } V \in \mathcal{R}. \end{cases}$$

The purpose of the dependency relation of a proof graph is to reflect a sound and complete reasoning for the truth of each of its nodes of the form $\langle \alpha, X, \bar{a}\rangle$, with X a variable defined in φ. Our definition of proof graph, below, will impose requirements on the dependency relation to ensure that the reflected reasoning is sound and complete. First, we present a local requirement on the dependency relation to ensure that the truth of a node $\langle \alpha, X, \bar{a}\rangle$ can, according to the definition of X in φ, be inferred from the truth of all its successors together. This local requirement should, moreover, be satisfied in every substructure with sufficient information from the perspective of the reflected reasoning. Thus, we obtain the intermediate notion of *dependency graph*, which only admits reasonings of which every individual inference step is justified. To obtain a suitable notion of proof graph, we then still need to add a global requirement that excludes non-wellfounded reasonings to the extent that they are invalid. We shall establish that the resulting notion of proof graph is sound and complete: all nodes in a proof graph for $\mathfrak{A}$, θ and φ are true in $\mathfrak{A}$ and θ, and if a node is true in $\mathfrak{A}$ and θ, then there is a proof graph that includes it.

Consider a node $v = \langle \alpha, X, \bar{a}\rangle$, with X a variable defined in φ, and let φ_X be the definition of X in φ. We denote by $v^\bullet$ the set of successors (the *postset*) of v. To express that if the elements of $v^\bullet$ are all true, then φ_X forces v to be true as well, we need to be able to influence the values of the second-order variables defined in φ_X. Let us denote by $\mathsf{fo}(\varphi_X)$ the formula obtained from φ_X by replacing every $[\mathbf{lfp}\, Y\bar{x}.\, \psi]\bar{t} \sqsubseteq \varphi_X$ by $Y\bar{t}$. We can then consider environments satisfying the nodes in $v^\bullet$, and require that such environments satisfy $\mathsf{fo}(\varphi_X)$ if, and only if, $\alpha = \mathbb{t}$. To avoid having to distinguish between the different values for α every time, we introduce the following shorthand notation:

$$\mathfrak{A}, \theta \overset{\alpha}{\models} \varphi \quad \text{denotes} \quad \mathfrak{A}, \theta \models \varphi \iff \alpha = \mathbb{t}.$$

Recall that we want to use proof graphs to define a notion of evidence for the validity or invalidity within a first-order structure $\mathfrak{A}$. Given a proof graph with nodes S, we will be looking at weak substructures of $\mathfrak{A}$ that have a domain of discourse that includes all the elements that are referenced by the nodes in S. We denote by $\mathfrak{A} \restriction S$ the smallest (with respect to $\sqsubseteq_{\mathrm{w}}$) weak substructure of $\mathfrak{A}$ of which the domain of discourse is a superset of $\{a \in A \mid \exists_{\langle \alpha, V, \bar{c}\rangle \in S}\ a \in \bar{c}\}$. Note that, according to the definition of weak substructure, if $\mathcal{I}$ is the interpretation function of $\mathfrak{A} \restriction S$, then $\mathcal{I}(R) = \emptyset$ for all first-order relation symbols R.

We enforce consistency of the reasoning represented by the graph by requiring that the successors of a node are never contradictory: a relation $\rightarrow \subseteq \mathcal{S} \times \mathcal{S}$ is *consistent* if and only if for all v, X and $\bar{a}$, not both $\langle \mathbb{t}, X, \bar{a}\rangle \in v^\bullet$ and $\langle \mathbb{f}, X, \bar{a}\rangle \in v^\bullet$.

▶ **Definition 3** (dependency graph)**.** A *dependency graph* for $\mathfrak{A}, \theta$ and φ is a directed graph $\langle S, \rightarrow\rangle$ with $S \subseteq \mathcal{S}$ and $\rightarrow \subseteq S \times S$, such that $\rightarrow$ is consistent, and for all $\langle \alpha, V, \bar{a}\rangle \in S$:

- if $V \in \mathsf{dv}(\varphi)$:

$$\text{for all } \mathfrak{A} \upharpoonright S \sqsubseteq_{\mathrm{w}} \mathfrak{B} \sqsubseteq_{\mathrm{w}} \mathfrak{A} \text{ and all } \eta \text{ such that } \eta \equiv_{\mathsf{fv}(\varphi)} \theta,$$
$$\text{if } \mathfrak{B}, \eta \models v \text{ for all } v \in \langle \alpha, V, \bar{a} \rangle^{\bullet} \text{ then } \mathfrak{B}, \eta[\bar{x}_V \mapsto \bar{a}] \stackrel{\alpha}{\models} \mathsf{fo}(\varphi_V)$$

- if $V \notin \mathsf{dv}(\varphi)$:

$$\mathfrak{A}, \theta \models \langle \alpha, V, \bar{a} \rangle \qquad \text{and} \qquad \langle \alpha, V, \bar{a} \rangle^{\bullet} = \emptyset$$

▶ **Example 4.** Let $\mathfrak{A}$ be the transition system in Figure 1. The left graph in Figure 3 is a dependency graph for $\mathfrak{A}$, any θ and φ_1, and the right graph in Figure 3 is a dependency graph for $\mathfrak{A}$, any θ and φ_2. Let us verify in some detail that the left graph in Figure 3 indeed satisfies the conditions of dependency graphs.

Clearly, the dependency relation of the graph in Figure 3 is consistent. Furthermore, the nodes $\langle \mathbb{t}, \xrightarrow{a}, [s_0, s_0] \rangle$ and $\langle \mathbb{t}, \xrightarrow{a}, [s_1, s_0] \rangle$ do not have successors and they are true in $\mathfrak{A}$.

For the nodes $\langle \mathbb{f}, X, [s_0] \rangle$ and $\langle \mathbb{f}, X, [s_1] \rangle$ we need to check the first condition of Definition 3. We show how to check the condition for $\langle \mathbb{f}, X, [s_0] \rangle$, the check for $\langle \mathbb{f}, X, [s_1] \rangle$ is similar. Consider a weak substructure $\mathfrak{B}$ of $\mathfrak{A}$ and an environment η ($\eta \equiv_{\mathsf{fv}(\varphi_1)} \theta$ holds for every η because $\mathsf{fv}(\varphi_1) = \emptyset$) such that $\mathfrak{B}, \eta \models \langle \mathbb{f}, X, [s_0] \rangle$ and $\mathfrak{B}, \eta \models \langle \mathbb{t}, \xrightarrow{a}, [s_0, s_0] \rangle$. The latter means that $\mathfrak{B}, \eta \models s_0 \xrightarrow{a} s_0$, and the former means that $\mathfrak{B}, \eta \not\models X s_0$, so $\mathfrak{B}, \eta \not\models s_0 \xrightarrow{a} s_0 \implies X s_0$. This is equivalent to $\mathfrak{B}, \eta^{\mathbb{f}} \models \mathsf{fo}(\varphi_X)$.

A least fixpoint is proved by an inductive argument, which is necessarily well-founded. A greatest fixpoint is proved by a coinductive argument, which need not be well-founded. In fact, a coinductive argument of a statement can be thought of as a reasoning that argues the absence of a well-founded reasoning for the negation of the statement. The quantification over all well-founded reasonings gives rise to a reasoning that is inherently not well-founded. A dependency graph admits reasonings that are not well-founded without taking into account whether such reasonings are justified. We formulate an extra requirement on dependency graphs to filter out invalid non-well-founded reasonings. Let $I_X(\pi)$ denotes the set of indices i on path π such that $\pi_i \in \mathcal{S}_{\{X\}}$.

▶ **Definition 5** (proof graph)**.** A dependency graph $G = \langle S, \to \rangle$ for $\mathfrak{A}, \theta$ and φ is a *proof graph* iff for every infinite path π in G, for all X minimal w.r.t. $<_\varphi$ such that $I_X(\pi)$ is infinite it holds that:

- if $\sigma_X = \mathbf{lfp}$, then $\{i \in I_X(\pi) \mid \exists_{\bar{a}}\ \pi_i = \langle \mathbb{t}, X, \bar{a} \rangle\}$ is finite; and
- if $\sigma_X = \mathbf{gfp}$, then $\{i \in I_X(\pi) \mid \exists_{\bar{a}}\ \pi_i = \langle \mathbb{f}, X, \bar{a} \rangle\}$ is finite.

▶ **Example 6.** As we saw in Example 4, the graphs in Figure 3 are dependency graphs; it remains to verify that they satisfy the condition of Definition 5 to conclude that they are proof graphs. For the proof graph on the right in Figure 3 the reasoning is as follows: On every infinite path π in the graph, each of the three nodes $\langle \mathbb{t}, X, [s_1] \rangle$, $\langle \mathbb{t}, Y, [s_1] \rangle$, and $\langle \mathbb{t}, Y, [s_2] \rangle$ occur infinitely often. Note that X is the (only) $<_{\varphi_2}$-minimal second-order variable on such an infinite path. We have that $\sigma_{\varphi_2, X} = \mathbf{gfp}$, and, indeed, the set $\{i \in I_X(\pi) \mid \exists_{\bar{a}}\ \pi_i = \langle \mathbb{f}, X, \bar{a} \rangle\}$ is finite.

The following theorem, which is proved for an equally expressive, but syntactically more general fixpoint logic (it has LFP as a normal form) in [6], establishes that the notion of proof graph defined above is sound and complete for capturing reasoning about the validity of relations defined by fixpoints in LFP formulas. Soundness here means that whenever a proof graph for a structure $\mathfrak{A}$, an environment θ and a formula φ includes a node $\langle \alpha, X, \bar{a} \rangle$ with X a second-order variable defined in φ, then $\langle \alpha, X, \bar{a} \rangle$ expresses a true statement with

respect to $\mathfrak{A}$, θ and φ. That is, the relation on $\mathfrak{A}$ associated with X by φ includes $\bar{a}$ if $\alpha = \mathbb{t}$, and the relation on $\mathfrak{A}$ associated with X by φ does not include $\bar{a}$ if $\alpha = \mathbb{f}$. Conversely, completeness means that if $\langle \alpha, X, \bar{a}\rangle$ expresses a true statement with respect to $\mathfrak{A}$, θ and φ, then there exists a proof graph for $\mathfrak{A}$, θ and φ including this node.

▶ **Theorem 7.** *Let $\mathfrak{A}$ be a first-order structure, let θ be an environment, let φ be an LFP formula, and let X be a second-order variable defined in φ. Then, for all $\langle \alpha, X, \bar{a}\rangle \in \mathcal{S}_{\{X\}}$, the following are equivalent:*

- *There exists a proof graph for $\mathfrak{A}$, θ and φ that includes the node $\langle \alpha, X, \bar{a}\rangle$.*
- $\bar{a} \in \sigma_X \mathbf{T}^{\mathfrak{A},\theta}_{\varphi_X, X, \bar{x}_X} \iff \alpha = \mathbb{t}$,

The notion of proof graph defined above deviates in two respects from the notion introduced for PBESs in [9]. Firstly, the new notion requires nodes to be included that capture the information about the first-order relations in the original structure that is needed in the reasoning reflected by the proof graph. These nodes will be used in Section 3.2 to extract an appropriate weak substructure that can serve as evidence. Secondly, the new notion includes an extra $\{\mathbb{t}, \mathbb{f}\}$-element in a node, which facilitates the inclusion in a proof graph of both positive and negative statements as to whether relations hold for certain sequences of elements, and an associated additional consistency requirement for the dependency relation. Such a facility was not needed in the setting of PBESs by the absence of negation.

3.2 Evidence

We proceed to define a general notion of evidence based on proof graphs. In Section 4 we illustrate how this notion can be used to provide diagnostics for behavioural equivalence checking, and we will show that this notion specialises to familiar notions of evidence in the context of model checking.

It follows from Theorem 7 that for LFP formulas of the form $[\sigma X\bar{x}.\,\varphi]\bar{t}$ we have that

$$\mathfrak{A}, \theta \overset{\alpha}{\models} [\sigma X\bar{x}.\,\varphi]\bar{t} \quad \text{iff} \quad \begin{array}{l}\text{there exists a proof graph for } \mathfrak{A}, \theta \text{ and } [\sigma X\bar{x}.\,\varphi]\bar{t} \\ \text{that includes a node } \langle \alpha, X, \bar{t}^{\mathfrak{A},\theta}\rangle.\end{array}$$

We shall refer to a proof graph for $\mathfrak{A}$, θ and $[\sigma X\bar{x}.\,\varphi]\bar{t}$ that includes the node $\langle \alpha, X, \bar{t}^{\mathfrak{A},\theta}\rangle$ as *a proof graph for* $\mathfrak{A}, \theta \overset{\alpha}{\models} [\sigma X\bar{x}.\,\varphi]\bar{t}$. Since, in fact, every LFP formula ψ is equivalent to $[\mathbf{lfp}\, X_0.\,\psi]$, provided that $X_0 \notin \mathsf{dv}(\psi) \cup \mathsf{fv}(\psi)$, we may use the terminology more generally: a proof graph for $\mathfrak{A}, \theta \overset{\alpha}{\models} \psi$ is a proof graph for $\mathfrak{A}, \theta \overset{\alpha}{\models} [\mathbf{lfp}\, X_0.\,\psi]$ if ψ is not already of the form $[\sigma X\bar{x}.\,\varphi]\bar{t}$ for some X, $\bar{x}$, φ and $\bar{t}$. We now propose the following definition of evidence for LFP formulas, which formalises that evidence based on some proof graph for the (in)validity of an LFP formula in a first-order structure is that part of the structure that facilitates the reasoning reflected by that proof graph.

▶ **Definition 8** (evidence)**.** By *evidence* for $\mathfrak{A}, \theta \overset{\alpha}{\models} \varphi$ we mean a weak substructure $\mathfrak{B} \sqsubseteq_{\mathrm{w}} \mathfrak{A}$ such that there is a proof graph for $\mathfrak{B}, \theta \overset{\alpha}{\models} \varphi$ that is also a proof graph for $\mathfrak{A}, \theta \overset{\alpha}{\models} \varphi$.

Naturally, we would like evidence to be as concise as possible, and so we would like to obtain, given a proof graph for $\mathfrak{A}, \theta \overset{\alpha}{\models} \varphi$, the smallest weak substructure of $\mathfrak{A}$ that serves as suitable evidence.

▶ **Definition 9** (evidence projection)**.** Given a proof graph $\langle S, \to\rangle$ for $\mathfrak{A}, \theta \overset{\alpha}{\models} \varphi$, define $\mathsf{ev}(\langle S, \to\rangle)$ as the smallest $\mathfrak{B} \sqsubseteq_{\mathrm{w}} \mathfrak{A}$ such that for each node $\langle \beta, V, \bar{a}\rangle \in S$ (with $V \in \mathcal{R} \cup \mathcal{X}$) we have $\bar{a} \in B^*$, and for each $v \in S \setminus \mathcal{S}_{\mathcal{X}}$ we have $\mathfrak{B}, \theta \models v$.

It is easy to see that $\mathsf{ev}(\langle S, \to \rangle)$ as defined in the preceding definition always exists, and that it can be computed straightforwardly from $\mathfrak{A} \restriction S$ by adding, for every node $\langle \mathbb{t}, R, \bar{a} \rangle$, the sequence $\bar{a}$ to the interpretation of relation symbol R.

▶ **Theorem 10.** *If G is a proof graph for $\mathfrak{A}, \theta \models^{\alpha} \varphi$, then $\mathsf{ev}(G)$ is evidence for $\mathfrak{A}, \theta \models^{\alpha} \varphi$.*

Our notion of evidence projection results in the smallest evidence for $\mathfrak{A}, \theta \models^{\alpha} \varphi$ given a particular proof graph. There may, however, be many proof graphs for $\mathfrak{A}, \theta \models^{\alpha} \varphi$, and it appears that, roughly speaking, smaller proof graphs lead to smaller evidence. A proof graph may contain redundant information; a subgraph that contains no redundant nodes or edges is called a *minimal* proof graph. We refer to [9] for a more elaborate discussion on minimality.

4 Counterexamples and witnesses

Some problems that can be encoded in fixpoint logic consist of checking an 'implementation' against a 'specification'. For instance, if the behaviour of some system is described as a Kripke structure, and we want to establish correctness properties on that Kripke structure, then we may view it as an 'implementation' of sorts, which we could check against a set of CTL* formulas, the 'specifications'. We might also want to check if this Kripke structure refines another, more abstract Kripke structure. In this case, the 'specification' is not a formula, but another Kripke structure. We refer to such problems as *model checking* problems.

For problems that have this characteristic of a division into implementation and specification, we tend to think of the specification as being given and well-understood, whereas the implementation may contain mistakes that need to be clarified with diagnostics. Such diagnostics should highlight the parts of the implementation that cause a problem, but should not include details from the specification. To achieve this, we propose a general scheme that combines an implementation $\mathfrak{A}$ with a specification $\mathfrak{B}$, and that extracts the information from $\mathfrak{A}$ from evidence relating to the combined structure. These combination and extraction operations are defined in terms of the operators $\cup$ and $\cap$. Essentially, the $\cup$ operator must merge two structures $\mathfrak{A}$ and $\mathfrak{B}$ together, and the $\cap$ operator must be able to retrieve a weak substructure of $\mathfrak{A}$ again from the merged structure. Natural candidates to implement these operations on the domain of discourse of the two structures are the set union and set intersection operations. The function and relation symbols are also obtained by taking the set union or intersection of the symbols in $\mathfrak{A}$ and $\mathfrak{B}$.

If R is a relation symbol with interpretations in both $\mathfrak{A}$ and $\mathfrak{B}$, then $R^{\mathfrak{A} \cup \mathfrak{B}} = R^{\mathfrak{A}} \cup R^{\mathfrak{B}}$ and $R^{\mathfrak{A} \cap \mathfrak{B}} = R^{\mathfrak{A}} \cap R^{\mathfrak{B}}$, and if R only has an interpretation in $\mathfrak{A}$ (resp. $\mathfrak{B}$), then $\mathcal{R}^{\mathfrak{A} \cup \mathfrak{B}} = \mathcal{R}^{\mathfrak{A}}$ (resp. $R^{\mathfrak{A} \cup \mathfrak{B}} = R^{\mathfrak{B}}$). A natural way to define the interpretation of a function symbol f in $\mathfrak{A} \cap \mathfrak{B}$ is to define $f^{\mathfrak{A} \cap \mathfrak{B}}$ as the restriction of $f^{\mathfrak{A}}$ to the new domain of discourse, $A \cap B$. Defining the interpretation for f in $\mathfrak{A} \cup \mathfrak{B}$ is problematic however, if $f^{\mathfrak{A}}$ and $f^{\mathfrak{B}}$ do not agree on the intersection of their domains. If they *do* agree on this intersection, we can define $f^{A \cup \mathfrak{B}}$ such that it assigns to every input the same output as $f^{\mathfrak{A}}$ does if the input is in the domain of $f^{\mathfrak{A}}$, or the output of $f^{\mathfrak{B}}$ if the input is in the domain of $f^{\mathfrak{B}}$.

For pairs of structures in which the interpretation of the function symbols are compatible in this way – we will call such structures *composable* – we define union and intersection operators $\cup$ and $\cap$ as described above. Using these operators, witnesses and counterexamples can be extracted as follows.

▶ **Definition 11.** If $\mathfrak{A}$ and $\mathfrak{B}$ are composable structures, $\mathfrak{E} \sqsubseteq_{\mathrm{w}} \mathfrak{A} \cup \mathfrak{B}$, θ is an environment and φ is an LFP formula such that $\mathfrak{E}$ is evidence for $\mathfrak{A} \cup \mathfrak{B}, \theta \models^{\alpha} \varphi$, then we call $\mathfrak{E} \cap \mathfrak{A}$ an $\mathfrak{A}$-*witness* if $\alpha = \mathbb{t}$. We call it an $\mathfrak{A}$-*counterexample* if $\alpha = \mathbb{f}$.

If, from the context, it is clear which structure was used to extract the $\mathfrak{A}$-witness, we simply refer to the resulting structure as a witness; likewise for counterexamples. A desirable property of a witness is that it can be related to the structure from which it is derived. Furthermore, a witness should contain all the information that is essential in proving the same LFP formula φ.

▶ **Theorem 12** ([6]). *If $\mathfrak{A}$ and $\mathfrak{B}$ are composable, θ is an environment, φ is an LFP formula over $\mathfrak{A} \cup \mathfrak{B}$, and $\mathfrak{C}$ is an $\mathfrak{A}$-counterexample or $\mathfrak{A}$-witness for $\mathfrak{A} \cup \mathfrak{B}, \theta \models^{\alpha} \varphi$, then*

$$\mathfrak{C} \sqsubseteq_w \mathfrak{A} \quad \text{and} \quad \mathfrak{C} \cup \mathfrak{B}, \theta \models^{\alpha} \varphi.$$

Usually, φ will be a closed formula, in which case the value of θ is irrelevant. In such cases, we will not explicitly mention θ, but assume that an arbitrary environment is given. In the following sections we will give an example of a formula that encodes stuttering equivalence checking, in which case $\mathfrak{A}$ and $\mathfrak{B}$ are Kripke structures, and an example of a formula that encodes ∃ECTL* model checking, in which case $\mathfrak{A}$ is a Kripke structure, and $\mathfrak{B}$ is a structure that represents an ∃ECTL* formula. This approach differs from those in [3, 14, 15], in which a different fixpoint formula is generated for every $\mathfrak{A}$ and $\mathfrak{B}$.

4.1 Counterexamples for stuttering bisimulation checking

To illustrate the use of the $\cup$ and $\cap$ operators on structures, and to illustrate how counterexamples can be extracted for an equivalence checking problem, we consider the problem of checking that two systems are stuttering bisimilar. We use Namjoshi's formulation of stuttering bisimulation [21], because it already closely resembles our definition in LFP.

▶ **Definition 13.** Given a Kripke structure $\langle A, AP, \to, \ell \rangle$, a relation $X \subseteq A \times A$ is a *stuttering bisimulation* if and only if it is symmetric, and there exist a well-founded order $\langle W, \prec \rangle$ and some mapping $rank : A \times A \times A \to W$ such that for all s, t such that Xst:

$$\begin{aligned} \ell(s) = \ell(t) \wedge \forall_u\ s \to u \implies & ((Xut \wedge rank(u,u,t) \prec rank(s,s,t)) \vee \\ & \exists_v\ t \to v \wedge ((Xsv \wedge rank(u,s,v) \prec rank(u,s,t)) \vee Xuv)). \end{aligned}$$

States s and t are said to be *stuttering bisimilar*, denoted $s \simeq t$, if a stuttering bisimulation exists that relates s and t.

▶ **Proposition 1** ([6]). *Let $\mathfrak{A}$ be a Kripke structure $\langle A, AP, \to, \ell \rangle$.*

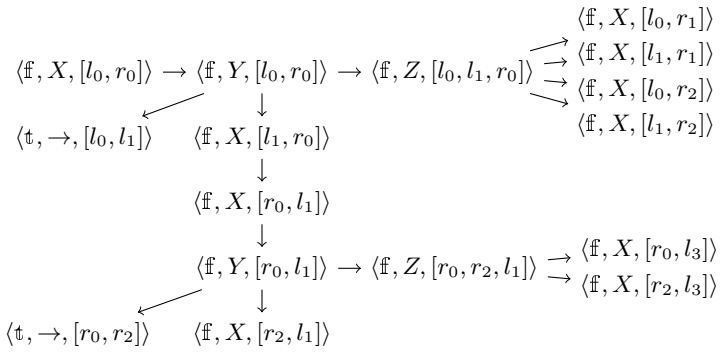
$$\begin{aligned} \Phi lr \triangleq\ & [\mathbf{gfp}\, Xst.\, Xts \wedge \ell(s) = \ell(t) \wedge \\ & [\mathbf{lfp}\, Yst.\, \forall_u\ s \to u \implies ((Xut \wedge Yut) \vee \\ & [\mathbf{lfp}\, Zsut.\, \exists_v\ t \to v \wedge ((Xsv \wedge Zsuv) \vee Xuv))]sut]st]lr \end{aligned}$$

If l and r are terms of $\mathfrak{A}$ and $s = l^{\mathfrak{A}}$ and $t = r^{\mathfrak{A}}$, then $\mathfrak{A} \models \Phi lr$ if and only if $s \simeq t$.

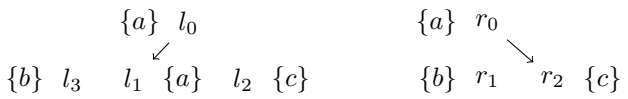
Consider the following two Kripke structures, that are stutter trace equivalent, but not stutter bisimulation equivalent.

$$L = \begin{array}{ccccc} & & \{a\} & l_0 & \\ & & \swarrow & & \searrow \\ \{b\} & l_3 \leftarrow & l_1 & \{a\} & l_2 \quad \{c\} \end{array} \qquad R = \begin{array}{ccccc} \{a\} & r_0 & & & \\ & \downarrow & & \searrow & \\ \{b\} & r_1 & & & r_2 \quad \{c\} \end{array}$$

Let $\mathfrak{A} = L \cup R$, and suppose that $l^L = l_0$ and $r^R = r_0$. Consider the following proof graph for $\mathfrak{A} \not\models \Phi lr$.



To extract evidence from this proof graph, we construct an evidence projection as per Definition 9. That is, we construct a substructure $\mathfrak{B} \sqsubseteq_{\mathrm{w}} \mathfrak{A}$ which must contain at least those states from L and R referred to in the proof graph (all states from L and R), and which satisfies $l_0 \to l_1$ and $r_0 \to r_2$. This yields the following Kripke structure $\mathfrak{B}$ as evidence. Note that $\mathfrak{B} \cap L$ and $\mathfrak{B} \cap R$ return the offending parts of L and R, respectively.



Observe that in $\mathfrak{B}$, l_0 and r_0 are again not stuttering bisimilar, and moreover, they can be shown not to be equivalent with the same reasoning: the transition from l_0 to a state unrelated to r_0 with label a cannot be mimicked by r_0. All the states from $\mathfrak{A}$ are retained in the evidence, because the existential quantifier requires an explanation for the invalidity of *every* X-node in the proof graph. Taking the projection of the proof graph minimised with respect to $\mathfrak{B}$ would yield evidence in which only the reachable states from l_0 and r_0 are included.

Other proof graphs are possible, leading to different evidence. For instance, if we had chosen $\langle \mathbb{F}, X, [l_0, r_0] \rangle$ to depend on $\langle \mathbb{F}, X, [r_0, l_0] \rangle$ (using the symmetry of stuttering bisimulation), we could have obtained evidence in which only the edge $r_0 \to r_1$ was retained. The explanation here is that it is sufficient to show that r_0 can reach an equivalence class labelled with b, without moving through another class first, whereas l_0 cannot do so.

We would like to remark that there are alternatives to our notion of evidence for bisimulation and stuttering bisimulation. For instance, a common notion is a distinguishing formula in Hennessy-Milner logic (for bisimulation [5]) or CTL*\X (for stuttering bisimulation [18]). However, in our experience, such formulas tend to get very unwieldy and do not always offer much insight. We believe that our notion of evidence can be a more practical alternative to distinguishing formulas in such cases.

4.2 Counterexamples for LTL and ACTL* model checking

In [4], Clarke et al. noted that for certain model checking problems, instead of generating substructures, one can generate counterexamples of a specific form: for LTL model checking, counterexamples are usually defined as a single (possibly infinite) trace through the model that does not satisfy the specification. These traces can again be seen as Kripke structures that do not satisfy the desired property. For model checking ∀CTL* – a subset of CTL* which adds to LTL universal quantification over branches – counterexamples consist of a number of traces that are attached to each other in a tree-like fashion. More formally, a tree-like counterexample is a Kripke structure that can be simulated by the system under

scrutiny, which does not satisfy the desired property, in which every strongly connected component (SCC) consists of a single cycle, and of which the SCC decomposition is a tree.

In the remainder of this section, we show that these special types of counterexample can be obtained from proof graphs. To simplify presentation, we consider the dual problem of generating witnesses for ∃CTL*. Furthermore, to also capture the expressivity of the ω-regular extensions used in [4], we consider the extended logic ∃ECTL* (originally presented in [25]), which uses Büchi automata as primitives. We note that it is also possible to define what follows directly for ∃CTL*, but this requires encoding the translation of LTL to Büchi automata in first-order logic, as was done in [8].

An ∃ECTL* formula f can be described by a structure $\mathfrak{B}_f$ over a domain that includes at least one element for every subformula and every set of subformulas of f, and for each subformula of the form $\mathsf{E}(\mathcal{B})$ a unique element for every state of $\mathcal{B}$. We let $\mathfrak{B}_f$ contain an element representing AP and an element representing the set F of accepting Büchi states. We assume that it also includes the usual relations on sets, relations to recover the structure of formulas, and a ternary transition relation $\rightarrow$ for the Büchi automata. To distinguish CTL* operators from Boolean connectives, we add a dot to the CTL* operators: $\dot\neg$ for CTL* negation, and $\dot\wedge, \dot\forall$ for CTL* conjunction and disjunction. A structured LFP encoding of the ∃ECTL* model checking problem is given below.

▶ **Proposition 2.** *Let Φ be defined as:*

$$\Phi sf \triangleq [\mathbf{lfp}\, Xsf.\ \begin{cases} (f \in AP \wedge f \in \ell(s)) \\ \vee\, \exists_g\ (f = \dot\neg g \wedge g \notin \ell(s)) \\ \vee\, \exists_{g,h}\ (f = g \dot\vee h \wedge (Xsg \vee Xsh)) \\ \vee\, \exists_{g,h}\ (f = g \dot\wedge h \wedge Xsg \wedge Xsh)) \\ \vee\, \exists_b\ (f = \mathsf{E}(\mathcal{B}(b)) \wedge \Psi sb) \end{cases}]sf$$

$$\Psi sb \triangleq [\mathbf{gfp}\, Ysb.$$
$$[\mathbf{lfp}\, Zsb. \exists_{s',b',g}\ s \rightarrow s' \wedge b \xrightarrow{g} b' \wedge Xsg \wedge$$
$$((b' \in F \wedge Ys'b') \vee (b' \notin F \wedge Zs'b'))]sb]sb$$

Let $\mathfrak{A}$ be a Kripke structure over AP, and let f be an ∃ECTL formula over AP. If s is a term of $\mathfrak{A}$ and f is a term of $\mathfrak{B}$, and $\hat a = s^{\mathfrak{A}}$ and $\hat b = f^{\mathfrak{B}_f}$, then $\mathfrak{A} \cup \mathfrak{B}_f \models \Phi sf$ if and only if $\mathfrak{A}, \hat a \models \hat b$.*

Let G be a minimal proof graph for $\mathfrak{A} \cup \mathfrak{B}_f \models \Phi sf$. Firstly, the first element of all nodes in G that are also in $\mathcal{S}_{\mathcal{X}}$ is equal to $\mathbb{t}$. Notice that per Definition 5, G cannot contain cycles that pass through $\mathcal{S}_{\{X\}}$. Because G is minimal, nodes from $\mathcal{S}_{\{Y,Z\}}$ have exactly one successor in $\mathcal{S}_{\{Y,Z\}}$. Therefore, the only cycles in G are cycles through $\mathcal{S}_{\{Y,Z\}}$, and every node in the proof graph can be in at most one cycle, leading to the following property:

▶ **Property 1.** *Let G be a minimal proof graph for $\mathfrak{A} \cup \mathfrak{B}_f \models \Phi sf$. Then every SCC in G consists of a single cycle.*

▶ **Example 14.** Consider the ∃ECTL* formula $\mathsf{E}(\mathcal{B}(b_0)) \dot\wedge \mathsf{E}(\mathcal{B}(b_1))$, where b_0 and b_1 are states of the Büchi automata below; the formula expresses that there are infinite y-paths and infinite x-paths.

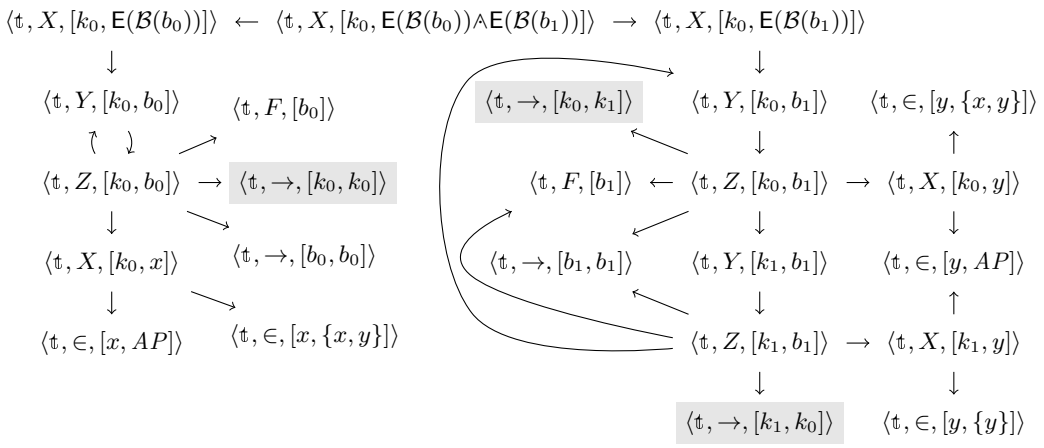


Figure 4 A proof graph explaining that state k_0 satisfies $\mathsf{E}(\mathcal{B}(b_0))\wedge\mathsf{E}(\mathcal{B}(b_1))$.

The state k_0 of the Kripke structure satisfies the ∃ECTL* formula. Following Proposition 2, $\Phi k_0(\mathsf{E}(\mathcal{B}(b_0))\wedge\mathsf{E}(\mathcal{B}(b_1)))$ must therefore hold. Indeed, a proof graph explaining this is given in Figure 4. This proof graph is minimal: none of its edges or nodes are redundant. The proof graph contains two SCCs, and no cycle passes through nodes from $\mathcal{S}_{\{X\}}$. The witness obtained from the proof graph of Figure 4, consisting of all Kripke structure states and relations defined by the shaded proof graph nodes, is essentially the original Kripke structure.

Our goal is to obtain a tree-like witness from a proof graph for $\mathfrak{A}\cup\mathfrak{B}_f \models \Phi sf$, if state $\hat{a}$ satisfies ∃ECTL* formula $\hat{b}$. We do so by first finding a witness in which every SCC is again a single cycle. We can however not use the witness obtained from G by Definition 11, because disjoint cycles in G may correspond to cycles in $\mathfrak{A}$ that share nodes. This may lead to a witness with SCCs that are no longer simple cycles.

To ensure that SCCs become simple cycles, in [4], cycles that share a subset of their nodes are 'unrolled'. In *ibid.* this is done by running a model checking algorithm not on $\mathfrak{A}$, but on a bisimilar *indexed Kripke structure* $\mathfrak{A}^\omega$, which contains for every cycle in $\mathfrak{A}$ an infinitely unrolled path. We adopt the same approach: we transform G to a proof graph for $\mathfrak{A}^\omega\cup\mathfrak{B}_f \models \Phi sf$, and then use Definition 11 to extract a witness from this proof graph.

▶ **Definition 15.** Given a Kripke structure $\mathfrak{A} = \langle A, AP, \to, \ell\rangle$, its corresponding *indexed Kripke structure* $\mathfrak{A}^\omega$ is the Kripke structure $\langle A^\omega, AP, \to^\omega, \ell^\omega\rangle$ such that:

- $A^\omega = A\times\mathbb{N}$,
- $\to^\omega$ is such that $\langle a,i\rangle \to^\omega \langle a',j\rangle$ iff $a\to a'$ (for all a, a', i and j),
- ℓ^ω is such that $\ell^\omega(\langle a,i\rangle) = \ell(a)$.

Note that every $a\in A$ is bisimilar to all $\langle a,i\rangle\in A^\omega$. Therefore, fixing some $i\in\mathbb{N}$ and replacing every $a\in A$ occurring as a parameter of a node in G by $\langle a,i\rangle$ yields a valid proof graph G^i (for $\mathfrak{A}^\omega\cup\mathfrak{B}_f$). For distinct i and j, the sets of nodes of G^i and G^j that have outgoing edges are disjoint, so $G^i\cup G^j$ is again a valid proof graph. By the same reasoning, so is $G^\omega = \bigcup_{i\in\mathbb{N}} G^i$. Associate with every node v of G a distinct number $k(v)$, fixing $k(\langle\mathbb{t}, X, [\hat{a},\hat{b}]\rangle) = 0$, and extend k to nodes of G^ω by defining for v in G and v' in G^ω that $k(v') = k(v)$ iff v' is equal to v in which every $a\in A$ is replaced by $\langle a,i\rangle$.

For every v in $G^\omega\cap\mathcal{S}_{\{Z\}}$, we replace every edge $v\to\langle\mathbb{t},V,[\langle a,i\rangle,b]\rangle$ with $V\in\{Y,Z\}$ and $i\neq k(v)$ by $v\to\langle\mathbb{t},V,[\langle a,k(v)\rangle,b]\rangle$. Note that v also has a successor $\langle\mathbb{t},\to,[\langle a',i\rangle,\langle a,i\rangle]\rangle$.

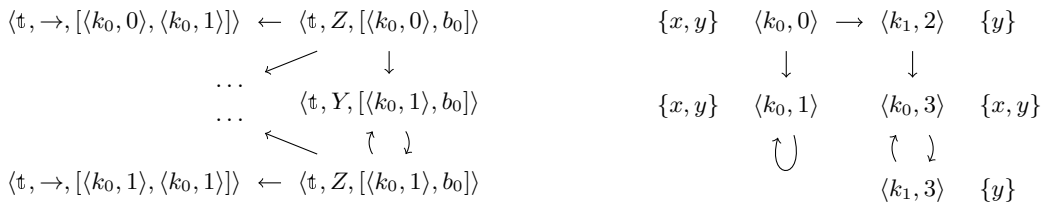
Replace this successor by the node $\langle \mathtt{t}, \to, [\langle a', i\rangle, \langle a, k(v)\rangle]\rangle$. Let G^{t} be the result of this transformation, restricted to the part that is reachable from $v_0 = \langle \mathtt{t}, X, [\langle \hat{a}, 0\rangle, \langle \hat{b}, 0\rangle]\rangle$. This transformation preserves Property 1:

▶ **Property 2.** *Every SCC in G^t consists of a single cycle.*

G^{t} is a valid dependency graph again; the restriction to the reachable part from v_0 is easily seen to preserve the conditions of Definitions 3 and 5. In the replacements we made, only the first and last conjunct in the right-hand side of the equation for Z are affected by a different choice for s'. These two conjuncts are represented by the new successors we introduced, satisfying the constraint from Definition 3.

To see that G^{t} is also a proof graph for $\mathfrak{A}^\omega \cup \mathfrak{B}_f \models \Phi s f$, notice that no 'bad' cycles were introduced during the transformation: if we view proof graphs as Kripke structures in which two nodes are labelled identically if and only if they differ only in the index of a state in $\mathfrak{A}^\omega$ (*i.e.*, if they are of the form $\langle \mathtt{t}, V, [\langle a, i\rangle, b]\rangle$ and $\langle \mathtt{t}, V, [\langle a, j\rangle, b]\rangle$), then all identically labelled nodes in G^ω are bisimilar. Moreover, we only replaced edges $v \to u$ by $v \to u'$ such that u and u' are bisimilar.

▶ **Example 16.** Consider the nodes $\langle \mathtt{t}, Z, [\langle k_0, j\rangle, b_0]\rangle$ in G^ω. Note that these have edges to $\langle \mathtt{t}, Y, [\langle k_0, j\rangle, b_0]\rangle$. Let $k(\langle \mathtt{t}, Z, [k_0, b_0]\rangle) = 1$. The transformation then yields edges $\langle \mathtt{t}, Z, [\langle k_0, j\rangle, b_0]\rangle \to \langle \mathtt{t}, Y, [\langle k_0, 1\rangle, b_0]\rangle$. Likewise, the successors $\langle \mathtt{t}, \to, [\langle k_0, j\rangle, \langle k_0, j\rangle]\rangle$ are all replaced with $\langle \mathtt{t}, \to, [\langle k_0, j\rangle, \langle k_0, 1\rangle]\rangle$; see the small snippet of G^{t} depicted below (left). A witness from G^{t}, as per Definition 11 is depicted below (right); note that it is tree-like.



We now define a witness $\mathfrak{C}$ as defined in Definition 11, *i.e.*, $\mathfrak{C} = \mathsf{ev}(G^{\mathrm{t}}) \cap \mathfrak{A}^\omega$. The following theorem characterises the shape of $\mathfrak{C}$; the theorem essentially follows from a correspondence between the transition relation of G^{t} and $\mathfrak{C}$.

▶ **Theorem 17** ([6]). *Every SCC in $\mathfrak{C}$ consists of a single cycle and $\mathfrak{C}$ is weakly connected.*

If the SCC decomposition of $\mathfrak{C}$ is not a tree, $\mathfrak{C}$ can be transformed to a bisimilar, tree-like structure $\mathfrak{C}^{\mathrm{t}}$ by duplicating SCCs with more than one incoming transition. $\mathfrak{A}$ simulates $\mathfrak{C}^{\mathrm{t}}$ because $\mathfrak{A}$ is bisimilar to $\mathfrak{A}^\omega$, $\mathfrak{C} \sqsubseteq_{\mathrm{w}} \mathfrak{A}^\omega$, and $\mathfrak{C}$ is again bisimilar to $\mathfrak{C}^{\mathrm{t}}$. The fact that f holds on $\mathfrak{C}$ follows from Proposition 2, so we may conclude that f also holds on the bisimilar $\mathfrak{C}^{\mathrm{t}}$.

▶ **Corollary 18.** *$\mathfrak{C}^t$ is a tree-like witness.*

In ∃ELTL model checking there is at most one Büchi automaton in the formula and therefore at most one cycle in G. The SCC duplication described above is then unnecessary.

▶ **Corollary 19.** *If f was an ∃ELTL formula, then $\mathfrak{C}$ is a linear witness.*

5 Concluding remarks

The diagnostics generation framework presented in this paper is inspired by Tan's attempt at diagnostics generation from *support sets* [23], which was shown to be flawed in [9]. The

diagnostics generation frameworks by Chechik and Gurfinkel [2], and Shoham and Grumberg [22] generate counterexamples and witnesses for CTL, for the purpose of counterexample guided abstraction refinement. The game-based approach of these frameworks is similar to ours, but aimed at a specific application, and in case of [22], at efficient computation. Our framework is more general in comparison, because it defines counterexamples and witnesses for a large variety of model checking problems, and also provides a notion of evidence for the more general setting of least fixpoint logic. Our contribution lies in providing a framework in which diagnostics for a wider variety of problems can be understood in the same way, while focusing less on how such diagnostics are obtained (although suggestions on how to do this are given in [6]).

In order to define our notion of evidence, we have adapted the notion of proof graph from [9] to include constructs that deal with negation. A side effect is that these proof graphs also induce a semantics for non-monotone formulas; for instance, one could assert that the formula $[\mathbf{lfp}\, Xn.\, n = 0 \vee \neg X(n-1)]4$ holds on the first-order structure $\langle \mathbb{Z}, -, 0 \rangle$, because there is a proof graph that witnesses it. It would be interesting to investigate whether this yields a usable semantics, and in particular, how it relates to the fixpoint theorem for non-monotonic functions in [10].

Our notion of evidence projection (see Definition 9) yields that part of a first-order structure that is relevant for the particular proof or refutation of a fixpoint formula captured by a proof graph. In some cases, the evidence projection alone will provide sufficient insight as to why the formula holds or does not hold, but in other cases it may be necessary to combine the evidence projection with additional information from the proof graph. We leave it as future work to further develop a theory of practical diagnostics based on the notions of proof graph and evidence discussed here.

Acknowledgements. The authors would like to thank the CSL reviewers for their constructive feedback and useful suggestions.

References

1 J. Brotherston and A. Simpson. Sequent calculi for induction and infinite descent. *J. Log. Comput.*, 21(6):1177–1216, 2011.

2 M. Chechik and A. Gurfinkel. A framework for counterexample generation and exploration. *STTT*, 9(5-6):429–445, 2007.

3 T. Chen, B. Ploeger, J.C. van de Pol, and T.A.C. Willemse. Equivalence checking for infinite systems using parameterized boolean equation systems. In *CONCUR 2007*, volume 4703 of *LNCS*, pages 120–135. Springer, 2007.

4 E.M. Clarke, S. Jha, Y. Lu, and H. Veith. Tree-like counterexamples in model checking. In *LICS 2002*, pages 19–29. IEEE Computer Society, 2002.

5 R. Cleaveland. On automatically explaining bisimulation inequivalence. In *CAV 1990*, volume 531 of *LNCS*, pages 364–372. Springer, 1991.

6 S. Cranen. *Getting the point – Obtaining and understanding fixpoints in model checking.* PhD thesis, Technische Universiteit Eindhoven, 2015. Available at `http://repository.tue.nl/791780`.

7 S. Cranen, J.F. Groote, J.J.A. Keiren, F.P.M. Stappers, E.P. de Vink, J.W. Wesselink, and T.A.C. Willemse. An overview of the mCRL2 toolset and its recent advances. In *TACAS 2013*, volume 7795 of *LNCS*, pages 199–213. Springer, 2013.

8 S. Cranen, J.F. Groote, and M.A. Reniers. A linear translation from CTL* to the first-order modal μ-calculus. *Theoretical Computer Science*, 412(28):3129–3139, 2011.

9 S. Cranen, B. Luttik, and T.A.C. Willemse. Proof graphs for parameterised boolean equation systems. In *CONCUR 2013*, volume 8052 of *LNCS*, pages 470–484. Springer, 2013.

10 Z. Ésik and P. Rondogiannis. A fixed point theorem for non-monotonic functions. *Theoretical Computer Science*, 574:18–38, 2015.

11 O. Friedmann and M. Lange. The modal μ-calculus caught off guard. In *TABLEAUX 2011*, volume 6793 of *LNCS*, pages 149–163. Springer, 2011.

12 C. Grabmayer. *Relating Proof Systems for Recursive Types*. PhD thesis, Vrije Univesiteit Amsterdam, 2005.

13 E. Grädel. Model checking games. *Electr. Notes Theor. Comput. Sci.*, 67:15–34, 2002.

14 J.F. Groote and R. Mateescu. Verification of temporal properties of processes in a setting with data. In *AMAST 1998*, volume 1548 of *LNCS*, pages 74–90. Springer, 1999.

15 J.F. Groote and T.A.C. Willemse. Model-checking processes with data. *Science of Computer Programming*, 56(3):251–273, 2005.

16 J.F. Groote and T.A.C. Willemse. Parameterised boolean equation systems. *Theoretical Computer Science*, 343(3):332–369, 2005.

17 D. Janin. Automata, tableaus and a reduction theorem for fixpoint calculi in arbitrary complete lattices. In *LICS 1997*, pages 172–182. IEEE Computer Society, 1997.

18 H. Korver. Computing distinguishing formulas for branching bisimulation. In *CAV 1991*, volume 575 of *LNCS*, pages 13–23. Springer, 1992.

19 L. Libkin. *Elements of Finite Model Theory*. Springer, 2004.

20 Y. Moschovakis. *Elementary induction on abstract structures*. North Holland, 1974.

21 K.S. Namjoshi. A simple characterization of stuttering bisimulation. In *FSTTCS 1997*, volume 1346 of *LNCS*, pages 284–296. Springer, 1997.

22 S. Shoham and O. Grumberg. A game-based framework for CTL counterexamples and 3-valued abstraction-refinement. *ACM Trans. Comput. Log.*, 9(1), 2007.

23 L. Tan. *Evidence-Based Verification*. PhD thesis, Department of Computer Science, State University of New York, 2002.

24 A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific journal of Mathematics*, 5(2):285–309, 1955.

25 W. Thomas. Computation tree logic and regular omega-languages. In *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, School/Workshop 1988*, volume 354 of *LNCS*, pages 690–713. Springer, 1989.

Elementary Elimination of Prenex Cuts in Disjunction-free Intuitionistic Logic*

Matthias Baaz[1] and Christian G. Fermüller[2]

1 **Institute of Discrete Mathematics and Geometry**
Vienna University of Technology, Austria
baaz@logic.at

2 **Theory and Logic Group 185.2**
Vienna University of Technology, Austria
chrisf@logic.at

Abstract

The size of shortest cut-free proofs of first-order formulas in intuitionistic sequent calculus is known to be non-elementary in the worst case in terms of the size of given sequent proofs with cuts of the same formulas. In contrast to that fact, we provide an elementary bound for the size of cut-free proofs for disjunction-free intuitionistic logic for the case where the cut-formulas of the original proof are prenex. To emphasize the non-triviality of our result, we establish non-elementary lower bounds for classical disjunction-free proofs with prenex cut-formulas and intuitionistic disjunction-free proofs with non-prenex cut-formulas.

1998 ACM Subject Classification F.4.1 Mathematical Logic (Proof Theory)

Keywords and phrases Cut-elimination, sequent calculus, intuitionistic logic

Digital Object Identifier 10.4230/LIPIcs.CSL.2015.94

1 Introduction

The elimination of cuts (*viz.* lemmas) from given sequent calculus proofs has remained in the focus of proof theory ever since Gentzen's seminal paper [4] from 1934/35. It is well known that the worst case complexity of cut-elimination is non-elementary for first-order intuitionistic as well as classical logic [8, 7, 3]. More precisely, there is a sequence of formulas $\langle F_i \rangle_{i\in\omega}$, where F_i has an **LI**-proof of length $\leq f(i)$, for some elementary function $f(i)$, while the shortest cut-free **LI**-proof of F_i is of length $\geq g(i)$ for some non-elementary function $g(i)$. (This means that $g(i)$ grows faster than $2^{\cdot^{\cdot^{\cdot^{2^i}}}}$, for any stack of 2s that is of constant height). Here **LI** is Gentzen's sequent calculus for first-order intuitionistic logic. The same result holds for the classical sequent calculus **LK**. The result is very robust: it does not matter whether we define the length of a proof as the number of symbols, formulas or just inference steps in it; moreover we may use any of the many known variants of Gentzen's original calculus.

It seems to be difficult to extract tight upper bounds from Gentzen's original cut-elimination procedure for **LI** [4]. Hudelmaier [5] provides a quadruple exponential upper bound for a suitable variant of *propositional* **LI**. However no elementary upper bound for cut-elimination seems to be known for non-trivial genuine first-order fragments of intuitionistic logics. The purpose of this paper is to show that one can in fact eliminate cuts involving

* This work was partly supported by Austrian Science Fund (FWF) projects P25417 and P26976.

only *prenex* cut-formulas from *disjunction-free* intuitionistic sequent proofs without non-elementary increase in the size of proofs. To obtain that elementary bound a new type of a cut-elimination argument is presented. The result is sharp in at least two respects. As we will show, in the following cases there exist non-elementary lower bounds for cut-elimination: (1) *classical* disjunction-free sequent derivations with prenex cut-formulas, and (2) intuitionistic disjunction-free sequent derivations with *non-prenex cut-formulas*.

The paper is organized as follows. In Section 2 we introduce the sequent calculus $\mathbf{LI}_m^{\neg\vee}$ for disjunction-free intuitionistic logic and fix corresponding terminology. We then present the overall procedure for eliminating prenex cut-formulas from $\mathbf{LI}_m^{\neg\vee}$-derivations in three steps. First, in Section 3, we consider the special case, where all cut-formulas in the given derivation are quantified atomic formulas. We then use this result in Section 4 in a transformation that trades arbitrarily complex prenex cut-formulas for propositional cut-formulas. In Section 5 we show how the remaining propositional cut-formulas can be eliminated from $\mathbf{LI}_m^{\neg\vee}$-derivations. In all three cases the depth of the resulting derivation will be elementarily bounded by the depth of the derivation we start with. In Section 6, we show that not only the depth, but also the size of the final cut-free proof is elementarily bounded in terms of the size of the original proof with arbitrary complex prenex cuts. While this follows straightforwardly for languages without function symbol, a further transformation step is involved in the presence of function symbols. In Section 7 we contrast the elementary upper bound for the elimination of prenex cuts with two cases where there exists a non-elementary lower bound for cut-elimination: disjunction-free classical logic with prenex atomic cuts and disjunction-free intuitionistic logic with non-prenex cuts.

2 Preliminaries

We work in a standard first-order language without identity. Terms are built up from constants and variables using function symbols, as usual. We follow Gentzen in syntactically distinguishing free variables $a_1, a_2, \ldots$ and bound variables $x_1, x_2, \ldots, y_1, \ldots$. Atomic formulas – *atoms*, for short – are of the form $P(t_1, \ldots, t_n)$, where P is a n-ary predicate symbol and $t_1, \ldots, t_n$ are terms. Formulas of (general) intuitionistic logic are built up from atoms using the propositional connectives $\neg$, $\wedge$, $\vee$, $\supset$ and the quantifiers $\forall$, $\exists$. If there are no occurrences of $\vee$ we speak of *disjunction-free* intuitionistic logic. The size $|F|$ of a formula F is the number of symbols occurring in it. A formula is *prenex* if it is of the form $\mathsf{Q}_1 x_1 \ldots \mathsf{Q}_n x_n A$, where A is propositional, ie., quantifier-free. If the quantifier-free part A of a prenex formula is an atom we speak of a *prenex atom*.

We consider the following variant of Gentzen's original calculus **LI** that we will refer to as $\mathbf{LI}_m^{\neg\vee}$. There is at most one formula at the right side of the sequent arrow, denoted here as $\vdash$; whereas on the left hand side we may have any finite multiset of formulas. In the following rules Γ and Π denote arbitrary finite multisets of formulas, Δ is either a single formula or empty. Multiset-union is denoted not by the comma, as usual. As usual, we write Γ, Π instead of $\Gamma \uplus \Pi$ and Γ, A instead of $\Gamma \uplus \{A\}$, etc, where $\uplus$ is the union operator for multisets.

Axioms:
$A \vdash A$ where A is atomic

Cut Rule:
$$\frac{\Gamma \vdash A \qquad A, \Pi \vdash \Delta}{\Gamma, \Pi \vdash \Delta}\ (cut)$$

Structural Rules:

$$\frac{\Gamma, A, A \vdash \Delta}{\Gamma, A \vdash \Delta}\,(contr) \qquad \frac{\Gamma \vdash \Delta}{A, \Gamma \vdash \Delta}\,(weak, l) \qquad \frac{\Gamma \vdash}{\Gamma \vdash A}\,(weak, r)$$

Logical Rules (Propostional and Quantifier Rules):

$$\frac{\Gamma, A \vdash}{\Gamma \vdash \neg A}\,(\neg, r) \quad \frac{\Gamma \vdash A}{\Gamma, \neg A \vdash}\,(\neg, l) \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \supset B}\,(\supset, r) \quad \frac{\Gamma \vdash A \quad B, \Pi \vdash \Delta}{\Gamma, \Pi, A \supset B \vdash \Delta}\,(\supset, l)$$

$$\frac{\Gamma \vdash A \quad \Pi \vdash B}{\Gamma, \Pi \vdash A \wedge B}\,(\wedge, r) \quad \frac{\Gamma, A \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta}\,(\wedge_1, l) \quad \frac{\Gamma, B \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta}\,(\wedge_2, l)$$

$$\frac{\Gamma \vdash A(a)}{\Gamma \vdash \forall x A(x)}\,(\forall, r) \quad \frac{\Gamma, A(t) \vdash \Delta}{\Gamma, \forall x A(x) \vdash \Delta}\,(\forall, l) \qquad \frac{\Gamma \vdash A(t)}{\Gamma \vdash \exists x A(x)}\,(\exists, r) \quad \frac{\Gamma, A(a) \vdash \Delta}{\Gamma, \exists x A(x) \vdash \Delta}\,(\exists, l)$$

In $(\forall, r)$ and $(\exists, l)$ a denotes an eigenvariable, i.e., a variable that does not occur in Γ. In $(\forall, l)$ and $(\exists, r)$ t denotes an arbitrary term.

Besides the missing rules for disjunction, $\mathbf{LI}_m^{-\vee}$ differs from Gentzen's original intuitionistic sequent calculus **LI** in the following respects: (1) $\mathbf{LI}_m^{-\vee}$ is based on multisets instead of Gentzen's sequences of formulas, which allows us to dispense with the exchange rule of **LI**. (2) Whereas Gentzen uses additive rules for introducing connectives, the logical rules of $\mathbf{LI}_m^{-\vee}$ are multiplicative, except for the left conjunction rules. The subscript m in $\mathbf{LI}_m^{-\vee}$ is intended as a reminder on this fact. (3) We insist on atomic axioms; i.e., axioms where the exhibited formula is atomic.

An $\mathbf{LI}_m^{-\vee}$-derivation of an *end-sequent* $\Gamma \vdash \Delta$ is an upward growing tree of sequents, obtained from instantiating the above rules as usual, starting with the root node $\Gamma \vdash \Delta$. If the end-sequent is $\vdash F$ we speak of a *proof of* F.

We need a few further technical notions for investigating $\mathbf{LI}_m^{-\vee}$-derivations. The exhibited formula occurrence (A) in the left and right upper sequents of the *cut*-rule is called *cut-formula*. The exhibited formula occurrence in the lower sequent of a structural, propositional or quantifier rule is called the *principal formula (occurrence)* of the corresponding inference. The formula occurrences exhibited in the upper sequent are called *immediate ancestors* of the corresponding principal formula in the lower sequent. The formulas in Γ, Π, Δ are called *side formulas*. Each occurrence of a side formula, say F, in the lower sequent of a rule has a unique corresponding occurrence of the same side formula F in (one of) the corresponding upper sequent(s). This upper occurrence of F is called the *immediate predecessor* of the lower occurrence of F.

Given a derivation γ, any occurrence F of a formula in γ spawns an *ancestor tree* $\tau_\gamma(F)$ defined inductively as follows:

- the given occurrence of F is the root of $\tau_\gamma(F)$;
- if G is a node in $\tau_\gamma(F)$, where G is principal formula of an inference in γ, then the immediate ancestor(s) of G in γ are (is) the successor node(s) of G;
- if G is a node in $\tau_\gamma(F)$, where G is a side formula of an inference in γ, then the immediate predecessor of G in γ is the successor node of G.

The *height* $h(\tau_\gamma(F))$ of the ancestor tree is defined as usual, where $h(\tau_\gamma(F)) = 0$ if $\tau_\gamma(F)$ consists only of the root F. Note that an ancestor tree only branches at nodes that are occurrences of principal formulas of contractions, i.e., applications of $(contr)$, or else of the rules $(\supset, l)$ or $(\wedge, r)$. Every leaf node of an ancestor tree occurs either in axiom or at the lower sequent of a weakening, i.e. an application of $(weak, l)$ or $(weak, r)$.

A derivation π is *regular* if each application of $(\forall, r)$ and $(\exists, l)$ in π is associated with a unique eigenvariable, which is converted into a bound variable by the corresponding inference.

A derivation π is *pruned* if every branch of π contains any sequent at most once and moreover each formula occurs at most three times on the left hand side of any sequent.

The size $|\pi|$ of a derivation π (in $\mathbf{LI}_m^{-\vee}$ or any other sequent calculus) is the sum of the sizes of all formulas occurrences in π. The height $h(\pi)$ of π is the largest number of consecutive inferences in π. (In other words, $h(\pi)$ is the maximal length of a branch of π.)

3 Prenex atomic cut-formulas

As outlined in the Introduction, we present the overall cut-elimination procedure in three stages. First, in this section, we consider the special case, where all cut-formulas in the given $\mathbf{LI}_m^{-\vee}$-derivation are quantified atoms.

We start with the following simple, but crucial observation.

▶ **Lemma 1.** *Let γ be a cut-free $\mathbf{LI}_m^{-\vee}$-derivation of $\Gamma \vdash A$, where A is a quantified atom $\mathsf{Q}\vec{x}P(\vec{t})$. Then the ancestor tree $\tau_\gamma(A)$ is not branching. I.e., $\tau_\gamma(A)$ consists in a unique thread of formula occurrences $A_1, \ldots, A_n$, where A_1 is the indicated occurrence of $A = \mathsf{Q}\vec{x}P(\vec{t})$ and A_{i+1} is either the immediate ancestor or the immediate predecessor of A_i in γ.*

Proof. It suffices to observe that all nodes in $\tau_\gamma(A)$ are quantified or unquantified atomic formulas that occur on the right hand side of a sequent. Therefore, each non-leaf node in $\tau_\gamma(A)$ has a unique successor that is either its immediate predecessor in γ or else the immediate ancestor of it for an inference $(\forall, r)$ or $(\exists, r)$. ◀

Remember that Gentzen [4] replaced the cut-rule by the mix-rule in order to formulation his argument for the eliminability of cuts. We will use a different generalization of (cut), called *multi-cut rule* (cut^+):

$$\frac{\Gamma_1 \vdash A_1 \quad \cdots \quad \Gamma_n \vdash A_n \qquad A_1, \ldots, A_n, \Pi \vdash \Delta}{\Gamma_1, \ldots, \Gamma_n, \Pi \vdash \Delta}\ (cut^+)$$

Clearly, for $n = 1$ (cut^+) coincides with the ordinary cut-rule (cut). On the other hand, (cut^+) is readily simulated by n applications of (cut). In the the following, we will assume that in $\mathbf{LI}_m^{-\vee}$ (cut) is replaced by by (cut^+).

We will call the rightmost upper sequent of an instance of (cut^+) its *main premise*. Let us call the occurrences of $A_1, \ldots, A_n$ in the main premise *lhs cut-formulas* (since they occur on the *l*eft *h*and *s*ide of the sequent). By the *lhs-depth* of an instance of the multi-cut rule in a derivation γ we mean the maximal height of the ancestor trees $\max(d(\tau_\gamma(A_1)), \ldots, d(\tau_\gamma(A_n)))$, where $A_1, \ldots, A_n$ are the lhs cut-formulas of this instance of (cut^+).

▶ **Theorem 2.** *Let π be an $\mathbf{LI}_m^{-\vee}$-derivation of $\Gamma \vdash \Delta$, where each cut-formula is a prenex atom. Then there exists an elementary function f, such that the following holds: there exists a cut-free $\mathbf{LI}_m^{-\vee}$-derivation π_0 of $\Gamma \vdash \Delta$, such that $h(\pi_0) \leq f(h(\pi))$.*

Proof. Throughout the proof we will assume implicitly that π is regular and that regularity is restored at each transformation step by using variable-renamed copies of sub-derivations where needed. We first focus on the elimination of (multi-)cuts and investigate the increase in complexity separately. Therefore we may assume without loss of generality that the last inference of π is the only instance of (cut^+) in π. In contrast to Gentzen's procedure (and its variants) we do not need nested induction in our case, but only induction over the lhs-depth d of the (only) multi-cut.

$d = 0$:
This entails that $n = 1$, since otherwise one of the lhs cut-formulas must have been introduced

already earlier in π (i.e., above the main premise), contradicting the assumption that the ancestor trees of the lhs cut-formulas consist of only those formulas themselves. There are two cases:

(1): If the main premise is an axiom then the application of the cut-rule is clearly redundant.

(2): If the cut-formula in the main premise has been introduced by weakening, then π has the form

$$\dfrac{\begin{array}{cc}\vdots & \dfrac{\begin{array}{c}\vdots\\ \Pi \vdash \Delta\end{array}}{A, \Pi \vdash \Delta}\,(weak, l)\\ \Gamma \vdash A & \end{array}}{\Gamma, \Pi \vdash \Delta}\,(cut^+)$$

and therefore the end-sequent $\Gamma, \Pi \vdash \Delta$ can be obtained without cut by continuing the upper right sub-derivation of $\Pi \vdash \Delta$ by iterated weakening to restore Γ.

$d > 0$:

We distinguish the following cases according to the type of the inference that has the main premise of the multi-cut as its lower sequent. We will refer to this inference as the 'relevant inference' in the following.

(1): If the principal formula of the relevant inference is not among the cut-formula the following subcases arise.

(1.1): The relevant inference is a unary propositional rule. We only present the case for $(\wedge_1, l)$, since $(\wedge_1, r)$, $(\supset, r)$, $(\neg, l)$, $(\neg, r)$ are treated analogously. π thus has the form

$$\dfrac{\begin{array}{cccc}\vdots & & \vdots & \dfrac{\begin{array}{c}\vdots\\ A_1, ..., A_n, C, \Pi \vdash \Delta\end{array}}{A_1, ..., A_n, C \wedge D, \Pi \vdash \Delta}\,(\wedge_1, l)\\ \Gamma_1 \vdash A_1 & \cdots & \Gamma_n \vdash A_n & \end{array}}{\Gamma_1, ..., \Gamma_n, C \wedge D, \Pi \vdash \Delta}\,(cut^+)$$

To decrease d, π is transformed into

$$\dfrac{\dfrac{\begin{array}{cccc}\vdots & & \vdots & \vdots\\ \Gamma_1 \vdash A_1 & \cdots & \Gamma_n \vdash A_n & A_1, ..., A_n, C, \Pi \vdash \Delta\end{array}}{\Gamma_1, ..., \Gamma_n, C, \Pi \vdash \Delta}\,(cut^+)}{\Gamma_1, ..., \Gamma_n, C \wedge D, \Pi \vdash \Delta}\,(\wedge_1, l)$$

(1.2): The relevant inference is a binary propositional rule, We present the case for $(\wedge, r)$; the case for $(\supset, l)$ is similar. π has the form

$$\dfrac{\begin{array}{cccc}\vdots & & \vdots & \dfrac{\begin{array}{cc}\vdots & \vdots\\ A_l^*, \Pi_l \vdash C & A_r^*, \Pi_r \vdash D\end{array}}{A_1, ..., A_n, \Pi \vdash C \wedge D}\,(\wedge, r)\\ \Gamma_1 \vdash A_1 & \cdots & \Gamma_n \vdash A_n & \end{array}}{\Gamma_1, ..., \Gamma_n, \Pi \vdash C \wedge D}\,(cut^+)$$

where $\Pi_l \uplus \Pi_r = \Pi$ and $A_l^* \uplus A_r^* = A_1, ..., A_n$, by which we mean that the multiset Π partitions into the disjoint sub-multisets Π_l and Π_r. Similarly $A_l^* = A_{i_1}, ...A_{i_k}$ and $A_r^* = A_{j_1}, ...A_{j_m}$, where $\{i_1, ...i_k\} \cup \{j_1, ...j_m\} = \{1, \ldots, n\}$ are disjoint subsets of indices.

Let γ_l denote the derivation

$$\dfrac{\begin{array}{cccc}\vdots & & \vdots & \vdots\\ \Gamma_{i_1} \vdash A_{i_1} & \cdots & \Gamma_{i_k} \vdash A_{i_k} & A_l^*, \Pi_l \vdash C\end{array}}{\Gamma_{i_1}, ..., \Gamma_{i_k}, \Pi_l \vdash C}\,(cut^+)$$

and let γ_r denote the derivation

$$\dfrac{\begin{array}{cccc}\vdots & & \vdots & \vdots\\ \Gamma_{j_1} \vdash A_{j_m} & \cdots & \Gamma_{j_1} \vdash A_{j_m} & A_r^*, \Pi_r \vdash C\end{array}}{\Gamma_{j_1}, ..., \Gamma_{j_m}, \Pi_r \vdash C}\ (cut^+)$$

Then π is replaced by

$$\dfrac{\begin{array}{cc}\gamma_l & \gamma_r\end{array}}{\Gamma_1, ..., \Gamma_n, \Pi \vdash C \wedge D}\ (\wedge, r)$$

(1.3): The relevant inference is a quantifier rule. We only present the case for $(\forall, r)$; the case for $(\exists, l)$ is analogous. The cases for $(\forall, l)$ and $(\exists, r)$ are similar, but even simpler, since they not involve variable renaming. For $(\forall, r)$ π has the form

$$\dfrac{\begin{array}{ccc}\vdots & & \vdots\\ \Gamma_1 \vdash A_1 & \cdots & \Gamma_n \vdash A_n\end{array} \quad \dfrac{\begin{array}{c}\vdots\,\delta\\ A_1, ..., A_n, \Pi \vdash F(a)\end{array}}{A_1, ..., A_n, \Pi \vdash \forall x F(x)}\ (\forall, r)}{\Gamma_1, ..., \Gamma_n, \Pi \vdash \forall x F(x)}\ (cut^+)$$

To decrease d, π is transformed into

$$\dfrac{\begin{array}{ccc}\vdots & & \vdots\\ \Gamma_1 \vdash A_1 & \cdots & \Gamma_n \vdash A_n\end{array} \quad \begin{array}{c}\vdots\,\delta'\\ A_1, ..., A_n, \Pi \vdash F(b)\end{array}}{\dfrac{\Gamma_1, ..., \Gamma_n, \Pi \vdash F(b)}{\Gamma_1, ..., \Gamma_n, \Pi \vdash \forall x F(x)}\ (\forall, r)}\ (cut^+)$$

where b is a free variable that does not occur in $\Gamma_1, ..., \Gamma_n$ and δ' arises from the δ by renaming a to b everywhere in this derivation.

(1.4): If the relevant inference is a structural rule, then the application of (cut^+) can straightforwardly be shifted upwards, similarly to the cases above.

(2): If the principal formula of the relevant inference is a cut-formula then the following sub-cases arise.

(2.1): If the relevant inference is a contraction $(contr)$ operating on one of the cut-formulas, then π has the form

$$\dfrac{\begin{array}{ccc}\vdots\,\delta & & \vdots\\ \Gamma_1 \vdash A_1 & \cdots & \Gamma_n \vdash A_n\end{array} \quad \dfrac{\begin{array}{c}\vdots\\ A_1, A_1, ..., A_n, \Pi \vdash \Delta\end{array}}{A_1, ..., A_n, \Pi \vdash \Delta}\ (contr)}{\Gamma_1, ..., \Gamma_n, \Pi \vdash \Delta}\ (cut^+)$$

To decrease d, π is transformed into

$$\dfrac{\begin{array}{cccccc}\vdots\,\delta' & \vdots & & \vdots & \vdots\\ \Gamma_1 \vdash A_1 & \Gamma_1 \vdash A_1 & \cdots & \Gamma_n \vdash A_n & A_1, A_1, ..., A_n, \Pi \vdash \Delta\end{array}}{\dfrac{\Gamma_1, \Gamma_1, ..., \Gamma_n, \Pi \vdash \Delta}{\Gamma_1, ..., \Gamma_n, \Pi \vdash \Delta}\ (contr)^*}\ (cut^+)$$

where $(contr)^*$ denotes a series of contractions and the sub-derivation δ' is obtained from δ by renaming free variables to ensure regularity (if needed).

(2.2): The relevant inference introduces a cut-formula by weakening. Then π has the form

$$\dfrac{\begin{array}{ccc}\vdots & & \vdots \\ \Gamma_1 \vdash A_1 & \cdots & \Gamma_n \vdash A_n\end{array} \quad \dfrac{\begin{array}{c}\vdots \\ A_2, ..., A_n, \Pi \vdash \Delta\end{array}}{A_1, A_2 ..., A_n, \Pi \vdash \Delta}\,(weak, l)}{\Gamma_1, ..., \Gamma_n, \Pi \vdash \Delta}\,(cut^+)$$

To decrease d, π is transformed into

$$\dfrac{\begin{array}{ccc}\vdots & & \vdots \\ \Gamma_2 \vdash A_2 & \cdots & \Gamma_n \vdash A_n\end{array} \quad \begin{array}{c}\vdots \\ A_2, ..., A_n, \Pi \vdash \Delta\end{array}}{\dfrac{\Gamma_2, ..., \Gamma_n, \Pi \vdash \Delta}{\Gamma_1, \Gamma_2, ..., \Gamma_n, \Pi \vdash \Delta}\,(weak, l)^*}\,(cut^+)$$

where $(weak, l)^*$ denotes a series of weakenings.

(2.3): The relevant inference introduces a quantifier of a cut-formula. We present the case for $(\exists, l)$. (The case for $(\forall, l)$ is similar.) Thus π has the form

$$\dfrac{\begin{array}{ccc}\vdots\delta & & \vdots \\ \Gamma_1 \vdash \exists x A(x) & \cdots & \Gamma_n \vdash A_n\end{array} \quad \dfrac{\begin{array}{c}\vdots\eta \\ A(a), A_2 ..., A_n, \Pi \vdash \Delta\end{array}}{\exists x A(x), A_2 ..., A_n, \Pi \vdash \Delta}\,(\forall, r)}{\Gamma_1, ..., \Gamma_n, \Pi \vdash \Delta}\,(cut^+)$$

where we first assume that the derivation δ has the following form:

$$\dfrac{\begin{array}{c}\vdots\gamma_0 \\ \Pi_1 \vdash A(t)\end{array}}{\Pi_1 \vdash \exists x A(x)}\,(\exists, r)$$

$$\gamma^- \begin{array}{c}\ddots\vdots \\ \Gamma_1 \vdash \exists x A(x)\end{array}$$

By Lemma 1, the ancestor tree of the occurrence of $\exists x A(x)$ in the end-sequent of δ consists in a single branch σ of formula occurrences. The indicated instance of $(\exists, r)$ denotes the location in σ, where $\exists x A(x)$ has $A(t)$ as its immediate ancestor; γ^- denotes the part of δ that is obtained by removing the sub-derivation γ_0 of $\Pi_1 \vdash \exists x A(x)$ from δ. The derivation replacing π, whereby d is decreased, can now be presented as

$$\dfrac{\begin{array}{ccc}\vdots\gamma_0 & & \vdots \\ \Pi_1 \vdash A(t) & \cdots & \Gamma_n \vdash A_n\end{array} \quad \begin{array}{c}\vdots\eta' \\ A(t), A_2 ..., A_n, \Pi \vdash \Delta\end{array}}{\Pi_1, ..., \Gamma_n, \Pi \vdash \Delta}\,(cut^+)$$

$$\gamma^+ \begin{array}{c}\ddots\vdots \\ \Gamma_1, ..., \Gamma_n, \Pi \vdash \Delta\end{array}$$

where η' is obtained from η by substituting all occurrences of a by t and γ^+ is obtained from γ^- by replacing the occurrences of $\exists x A(x)$ in the ancestor tree σ by Δ and additionally adding $\Gamma_2, \ldots, \Gamma_n, \Pi$ at the left hand side of those sequents where Δ has replaced $\exists x A(x)$.

The other subcase arises if the uppermost occurrence of $\exists x A(x)$ in the derivation δ is not introduced by an application of $(\exists, r)$, as above, but by weakening. In other words γ_0 ends in $\Pi_1 \vdash$. This derivation leads to a cut-free derivation of $\Gamma_1 \vdash$ and therefore also one of $\Gamma_1, ..., \Gamma_n, \Pi \vdash \Delta$ by iterated weakening.

This concludes the description of steps for shifting and eventually eliminating an uppermost cut. By repeating the argument, we arrive at the desired cut-free proof π_0 of $\Gamma \vdash \Delta$.

As for the complexity bound, we first investigate the increase in the height of the derivation for the elimination of a single cut. To this aim, note that for any sequent in the original derivation π the maximal number of occurrences of formulas at the left hand side is not increased throughout the cut-elimination procedure. This number is clearly exponentially bounded in terms of $h(\pi)$. Next we look at single steps of moving the cut upwards in the derivation, as indicated in the various cases of the inductive proof. No increase in height arises at the first base case ($d = 0$, cut with an axiom). For the other base case ($d = 0$, cut with a cut-formula introduced by $(weak, l)$), the weakenings needed to turn $\Gamma \vdash \Delta$ into $\Gamma, \Pi \vdash \Delta$ in the new derivation without cut may increase its height by at most h. For the cases where $d > 0$, we observe that in the cases (1.1), (1.2), (1.3), and (1.4) the height of the derivation is increased at most by 1. In case (2.1) the height may be increased by the additional applications of $(contr)$; in case (2.2) the height may be increased by the additional applications of $(weak, l)$; in case (2.3) the height may be increased by appending the derivation γ^+ below the cut. In all three cases the increase is again bounded by h. Since the lhs-depth d is decreased at each step and since $d \leq h$, we arrive at a bound h^2 for the height of a derivation where an uppermost cut in π is eliminated. In repeating the argument for the next cut to be eliminated, we have to replace h by h^2, resulting in the bound $h^{2^2} = h^4$. Continuing in this manner until the last cut is eliminated, we obtain that the height $h(\pi_0)$ of the final derivation π_0 is bounded by $h^{(2^h)}$, which is clearly elementary in $h(\pi)$. ◀

▶ Remark. Clearly tighter complexity bounds could be extracted. However we are only interested in the contrast between an elementary and a non-elementary increase, here.

▶ Remark. Our argument essentially differs in several respects from other cut-elimination proofs, in particular also from Gentzen's original [4]. That unnested induction over the lhs-depth of a (multi-)cut suffices in our case is due to the observation stated as Lemma 1. It only holds in the absence of disjunction.

4 Complex prenex cut-formulas

In Section 3 we have seen that cuts with prenex *atomic* cut-formulas can be eliminated from $\mathbf{LI}_m^{\neg\vee}$-derivations without incurring a non-elementary increase in the height of proofs. In this section we show that arbitrary complex prenex cut-formulas can be replaced by atomic ones for the price of introducing propositional cuts, i.e., applications of *cut*, where the cut-formula is quantifier free. The complexity of this transformation is negligible in our context.

▶ **Theorem 3.** *Let π be an $\mathbf{LI}_m^{\neg\vee}$-derivation of $\Gamma \vdash \Delta$, where each cut-formula is prenex. Then there exists an $\mathbf{LI}_m^{\neg\vee}$-derivation π_0^p of $\Gamma \vdash \Delta$ that only contains propositional cut-formulas, such that $h(\pi_0^p) \leq f(h(\pi))$ for some elementary function f.*

Proof. We first present the transformation of π into π_0^p in three separate stages and investigate the corresponding increase in height afterwards. (Again, regularity is assumed without further mentioning throughout the proof.)

Stage 1: We first consider the case, where the last inference of π is the only cut in π. The case where the cut-formula is a prenex atom is covered by Theorem 2. Therefore we assume that the cut-formula is of the form $\vec{\mathsf{Q}}\vec{x}B$, where B is a compound propositional formula and $\vec{\mathsf{Q}}\vec{x}$ denotes a non-empty string $\mathsf{Q}_1 x_1 \dots \mathsf{Q}_n x_n$ of quantifier occurrences, where $\mathsf{Q}_i \in \{\forall, \exists\}$ for $1 \leq i \leq n$. We first make the following observation.

▶ **Fact 1.** *If either the ancestor tree of the left cut-formula in π or the ancestor tree of the right cut-formula in π only contains quantified formulas, then the corresponding cut-formula can be traced back to ancestors that have been introduced by weakening. In that case the cut can be eliminated just like in the analogous cases in the proof of Theorem 2.*

In the remaining (general) case π can be depicted as follows.

$$\dfrac{\dfrac{\vdots}{\Gamma' \vdash \rho B}\,(\mathsf{Q}_n, r) \qquad \dfrac{\vdots}{\sigma B, \Pi' \vdash \Delta'}\,(\mathsf{Q}_n, l)}{\begin{array}{cc}\ddots & \vdots \\ \Gamma \vdash \vec{\mathsf{Q}}\vec{x}B & \end{array} \qquad \begin{array}{cc} & \vdots \quad \iddots \\ \vec{\mathsf{Q}}\vec{x}B, \Pi \vdash \Delta & \end{array}}\,(cut)$$
$$\Gamma, \Pi \vdash \Delta$$

where ρB denotes a quantifier-free formula occurrence in the ancestor tree of the cut-formula at the left premise of (cut), where (Q_n, r) introduces the innermost quantifier $\mathsf{Q}_n x_n$ of $\vec{\mathsf{Q}}\vec{x}B$. Analogously, for σB and (Q_n, l) at the right part of the derivation. ρ and σ are the substitutions that replace the bound variables in B by appropriate free variables or terms, respectively.

Let $A = P_B(a_1, \ldots, a_n, b_1, \ldots, b_m)$ be an atom, where P_B is a new predicate symbol, $a_1, \ldots, a_n$ are free variables corresponding to the bound variables occurring in B, and $b_1, \ldots, b_m$ are free variables corresponding to the free variables occurring in B. We then introduce implications that define A as 'abbreviation' of B. Accordingly π is transformed into the following derivation π'.

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\Pi \vdash \rho B} \quad \dfrac{\vdots}{\rho A \vdash \rho A}\,(\supset, l)}{\rho B \supset \rho A, \Pi \vdash \rho A}\,(\forall, l)^*}{\forall \vec{x}(B \supset A), \Pi \vdash \rho A}\,(\mathsf{Q}_1, r) \qquad \dfrac{\dfrac{\dfrac{\vdots}{\Pi \vdash \sigma A} \quad \dfrac{\vdots}{\sigma B \vdash \sigma B}\,(\supset, l)}{\sigma A, \sigma A \supset \sigma B, \Pi' \vdash \Delta'}\,(\forall, l)^*}{\sigma A, \forall \vec{x}(A \supset B), \Pi' \vdash \Delta'}\,(\mathsf{Q}_1, l)}{\begin{array}{cc}\ddots & \vdots \\ \forall \vec{x}(B \supset A), \Gamma \vdash \vec{\mathsf{Q}}\vec{x}A & \end{array} \qquad \begin{array}{cc} & \vdots \quad \iddots \\ \vec{\mathsf{Q}}\vec{x}A, \forall \vec{x}(A \supset B), \Pi \vdash \Delta & \end{array}}\,(cut)$$
$$\forall \vec{x}(B \supset A), \forall \vec{x}(A \supset B), \Gamma, \Pi \vdash \Delta$$

This transformation is repeated for every cut-formula that is not already a prenex atomic formula.

Stage 2: The derivation π' obtained from Stage 1 contains only prenex atomic cut-formulas. Therefore we can apply Theorem 2 to obtain a cut-free derivation π'_0 of $\Pi, \Gamma \vdash \Delta$, where Π denotes the 'defining implications' introduced in Stage 1.

Stage 3: For every cut-formula $\vec{\mathsf{Q}}\vec{x}B$ of the original derivation π, we replace all occurrences of (instances of) the atoms A, introduced in Stage 1, by (corresponding instances of) B. This results in sub-derivations that have the following form.

$$\dfrac{\dfrac{\vdots}{\Psi_l \vdash \rho B} \qquad \dfrac{\vdots}{\rho B, \Psi_r \vdash \Lambda'}\,(\supset, l)}{\rho B \supset \rho B, \Psi_l, \Psi_r \vdash \Lambda'}$$
$$\begin{array}{ccc}\ddots & \vdots & \iddots \\ \forall \vec{x}(B \supset B), \Psi_l, & \Psi_r, & \Psi \vdash \Lambda\end{array}$$

The indicated instance of $(\supset, l)$ can be replaced by an instance of (cut) to obtain

$$\dfrac{\dfrac{\vdots}{\Psi_l \vdash \rho B} \qquad \dfrac{\vdots}{\rho B, \Psi_r \vdash \Lambda'}}{\Psi_l, \Psi_r \vdash \Lambda'}\,(cut)$$

$$\ddots \qquad \vdots \qquad \iddots$$

$$\Psi_l, \Psi_r, \Psi \vdash \Lambda$$

Note that the replacement of the atom A by the compound formula B actually results in a derivation that is not a proper $\mathbf{LI}_m^{-\vee}$-derivation, since it will contain leaf nodes of the form $C \vdash C$, where C is a compound propositional formula. However such 'improper axioms' can readily be replaced by derivations from atomic axioms. This finally results in the derivation π_0^p of $\Gamma \vdash \Delta$ that trades prenex atomic cuts for propositional cuts.

It remains to investigate the increase in the height of the derivation. In Stage 1, in the case covered by Fact 1 we argue like in the corresponding case in the proof of Theorem 2: a cut-formula introduced by weakening is eliminated for the (possible) price of at most h additional weakenings, where $h = h(\pi) + 1$. In the general case, exhibited above, the transformation in Stage 1 may increase the depth by the additional introductions of universal quantifiers as indicated. There are as many such inferences as there are quantifier occurrences in the corresponding cut-formula. But these quantifier occurrences have been introduced by corresponding instances of quantifier rules already in the original derivation π and therefore the increase in height is again bounded by h. Repeating this for all cut-formulas, we obtain an overall bound of h^2 for Stage 1.

For Stage 2 we obtain an elementary bound from Theorem 2.

In Stage 3 the increase in height arises from the augmented derivations of improper (non-atomic, but propositional) axioms $C \vdash C$. The height of such derivations is bounded by the size of C, which in turn is not greater than $h(\pi)$, since C must already have been introduced from (atomic) axioms in π. (Remember that we have eliminated cut-formulas that trace back to ancestors introduced by weakening.) Moreover there are at most $h(\pi)$ such formulas.

Summing up, we obtain that the height of the final derivation π_0^p is elementarily bounded in the height of the original derivation π. ◀

5 Eliminating propositional cuts

We complete our cut-elimination proof by showing that propositional cuts can always be eliminated from $\mathbf{LI}_m^{-\vee}$-derivations without incurring a non-elementary increase in proof size.

▶ **Theorem 4.** *Let π be an $\mathbf{LI}_m^{-\vee}$-derivation of $\Gamma \vdash \Delta$, where each cut-formula is propositional. Then there exists a cut-free $\mathbf{LI}_m^{-\vee}$-derivation π_0 of $\Gamma \vdash \Delta$, such that $h(\pi_0) \leq f(h(\pi))$ for some elementary function f.*

Proof. We will not directly argue about the (ordinary) height $h(\pi)$ of a derivation π, but instead consider the variant $\bar{h}(\pi)$, defined like $h(\pi)$, except that applications of the contraction and weakening rules are not counted. In other words $\bar{h}(\pi)$ is the maximal number of applications propositional and quantifier rules occurring in any branch of π. We will also talk of the *depth of a formula occurrence* F in a derivation π. By this we mean the height $\bar{h}(\pi')$ without counting contractions and weakenings in the sub-derivation π' of π that has as its root the sequent containing the indicated formula occurrence F.

Let $CF(\pi)$ denote the set of different propositional formulas that occur as cut-formulas in π, augmented by all their subformulas. Starting with a cut-formula F of maximal size we will stepwise reduce $CF(\pi)$ until it is empty. The following cases arise.

(1) F is atomic: We consider all sub-derivations of π that end in a cut, where the atomic cut-formula F is a deepest occurrence of F in π. Let

$$\dfrac{\begin{array}{c}\vdots\,\delta\\ \Gamma \vdash F\end{array} \qquad \begin{array}{c}F \vdash F \quad \cdots \quad F \vdash F\\ \ddots \quad \vdots\,\gamma \quad \iddots\\ F, \Pi \vdash \Delta\end{array}}{\Gamma, \Pi \vdash \Delta}\,(cut)$$

be such a sub-derivation, where the exhibited axioms in γ are those where the occurrence of F at the left side is in the ancestor tree of the exhibited cut-formula F. (Note that, by assumption, neither δ nor γ contains a cut with F.) The cut is eliminated by replacing theses axioms with copies of the sub-derivation δ, resulting in

$$\dfrac{\begin{array}{c}\vdots\,\delta\\ \Gamma \vdash F\end{array} \quad \cdots \quad \begin{array}{c}\vdots\,\delta\\ \Gamma \vdash F\end{array}}{\Gamma, \ldots, \Gamma, \Pi \vdash \Delta}\,(contr)^*$$

The above picture is in fact only adequate if all leaf nodes of the mentioned ancestor tree of the cut-formula F occur in axioms. But such ancestors of the cut-formula may also result from applications of $(weak, l)$. In each such case the introduction of the corresponding occurrence of F is redundant, which in turn might render also applications of the contraction rule that lead to the cut-formula itself redundant. In fact it can happen that there is no axiom at all that contains an ancestor of the cut-formula. In that case, the cut, together with corresponding applications of $(weak, l)$ and possibly also $(contr)$ is redundant as well.

Another processing step that is left implicit in the above picture is the restoration of regularity to ensure that the eigenvariable condition for quantifier rules is preserved in further transformation steps. This is readily achieved by using variable-renamed copies of the sub-derivation δ.

We additionally eliminate other cuts with the cut-formula F occurring γ in the same manner, i.e., by replacing the axioms that contain the leaf nodes of the ancestor tree of the cut-formula F at the right upper sequent of the corresponding cut by δ (and/or eliminating redundant applications of weakening and contraction). While this renders the left upper sequent of the cut – say $\Pi' \vdash F$ – redundant we have to add the missing multiset Π' of formulas by applying additional weakenings at the position, where originally the cut-rule was applied.

As already indicated, the above transformation is to be applied simultaneously to all deepest occurrences of the atomic formula F as cut-formula in π. There might be further occurrences of F as cut-formulas in the resulting derivation π', entailing $CF(\pi') = CF(\pi)$. However note that each such occurrence of F must be less deep in π, than the deepest occurrences of F as cut-formula in π. Furthermore note that $\bar{h}(\pi') \leq \bar{h}(\pi) + \bar{h}(\delta)$. By iterating the transformation (always applied to all currently deepest occurrence of the cut-formula F) we arrive at a derivation π'', where $CF(\pi'') = CF(\pi) - \{F\}$ and $\bar{h}(\pi'') \leq \bar{h}(\pi) \cdot \bar{h}(\delta) \leq \bar{h}(\pi)^2$.

(2) ***F* is not atomic:** We only consider the case, where F is of the form $A \supset B$; the cases for negation and conjunction are similar. We depict π as follows:

$$\dfrac{\dfrac{\vdots}{\Gamma_0, A \vdash B}}{\Gamma_0 \vdash A \supset B}\,(\supset, r) \quad \dfrac{\vdots\gamma_1^A \quad \vdots\gamma_1^B}{\Pi_1' \vdash A \quad B, \Pi_1'' \vdash \Delta_1}\,(\supset, l) \quad \cdots \quad \dfrac{\vdots\gamma_n^A \quad \vdots\gamma_n^B}{\Pi_n' \vdash A \quad B, \Pi_n'' \vdash \Delta_n}\,(\supset, l)$$

$$\dfrac{\ddots\ \overset{\delta}{\vdots} \qquad\qquad \ddots\ \overset{\gamma}{\vdots}\ \iddots}{\Gamma, \Pi \vdash \Delta}\,(cut)$$

where $\Pi_i = \Pi_i' \uplus \Pi_i''$ for $i \in \{1, \ldots, n\}$. We first assume that no occurrence of $A \supset B$ in the ancestor trees of the two cut-formulas is introduced by weakening. The exhibited occurrences of $(\supset, r)$ and of $(\supset, l)$ indicate all locations in π, where the first occurrence of the cut-formula in the corresponding ancestor tree is introduced. Let δ' denote the derivation ending in $A, \Gamma \vdash B$ that results from δ by eliminating the exhibited occurrences of $(\supset, r)$. The exhibited cut is eliminated by transforming π into

$$\dfrac{\dfrac{\dfrac{\vdots\gamma_1^A \quad \vdots\delta'}{\Pi_1' \vdash A \quad A, \Gamma \vdash B}\,(cut) \qquad \vdots\gamma_1^B}{\Gamma, \Pi_1' \vdash B \qquad\qquad B, \Pi_1'' \vdash \Delta_1}\,(cut)}{\Gamma, \Pi_1 \vdash \Delta_1}\,(cut) \quad \cdots \quad \dfrac{\dfrac{\dfrac{\vdots\gamma_n^A \quad \vdots\delta'}{\Pi_n' \vdash A \quad A, \Gamma \vdash B}\,(cut) \qquad \vdots\gamma_n^B}{\Gamma, \Pi_n' \vdash B \qquad\qquad B, \Pi_n'' \vdash \Delta_1}\,(cut)}{\Gamma, \Pi_n \vdash \Delta_n}\,(cut)$$

$$\dfrac{\ddots\ \vdots\ \iddots}{\Gamma, \ldots, \Gamma, \Pi \vdash \Delta}\,(contr)^*$$

where the various copies of the sub-derivation δ' are variable-renamed to ensure the eigenvariable condition. If an occurrence of $A \supset B$ in the ancestor tree of one of the two cut-formulas is introduced by weakening, rather than by an implication rule, then the cut can be eliminated as usual at the possible expense of additional weakening to keep the side formulas of the redundant upper sequents of the cut in the derivation.
Like in case (1) we also eliminate all other cuts with cut-formula $A \supset B$ that occur in γ (i.e., above the right upper sequent of the exhibited deepest cut in π) in an analogous manner to obtain a derivation π', where the deepest occurrence of $A \supset B$ as cut-formula is smaller than in π. Also like in case (1), we have $\bar{h}(\pi') \leq \bar{h}(\pi) + \bar{h}(\delta)$ and, by iterating the transformation, obtain a derivation π'', where $CF(\pi'') = CF(\pi) - \{A \supset B\}$ and $\bar{h}(\pi'') \leq \bar{h}(\pi) \cdot \bar{h}(\delta) \leq \bar{h}(\pi)^2 \leq h(\pi)^2$.

We have seen that the elimination of a single formula from $CF(\pi)$ may be achieved at a quadratic expense in terms of $h = h(\pi)$. Eliminating a second cut-formula from $CF(\pi)$ therefore results in a derivation of height $\leq (h^2)^2 = h^4$. By repeating the transformation for all n formulas in $CF(\pi)$ we thus obtain the bound $h^{(2^n)}$ for a cut-free proof π_0' of the original end-sequent $\Gamma \vdash \Delta$. Since $n \leq 2^h$, we have $\bar{h}(\pi_0') \leq g(h(\pi))$ for some elementary function g.

Note that in extracting this bound from our cut-elimination procedure we made essential use of the fact that applications of the contraction and weakening rules are not counted in $\bar{h}(\pi_0')$. To establish an elementary bound in terms of the ordinary height, as stated in the theorem, we finally have to remove redundant copies of sequents as well as redundant formula occurrences from sequents in π_0'. More precisely, we prune π_0' to obtain π_0 as follows. (A similar pruning process is already implicit in [4] and described in more detail in [3].)

(1) Suppose π_0' contains a redundant copy of a sequent $\Psi \vdash \Lambda$. Then we replace

$$\begin{array}{c} \vdots\gamma \\ \Psi \vdash \Lambda \\ \ddots\vdots\iddots \\ \Psi \vdash \Lambda \\ \ddots\vdots\iddots \\ \Gamma \vdash \Delta \end{array}$$

by

$$\begin{array}{c} \vdots\gamma \\ \Psi \vdash \Lambda \\ \ddots\vdots\iddots \\ \Gamma \vdash \Delta \end{array}$$

(2) To ensure that each sequent contains at most three occurrences of the same formula at the left hand side of any sequent, we proceed as follows. Tracing the derivation downwards from the axioms, we apply the contraction rule immediately below any sequent containing two copies of the same formula at the left hand side. (More than one such application of (*contr*) may be needed, since a binary inference rule may result in several pairs of copies of the same formula.) This leaves us with a derivation, where a formula A may disappear altogether from a lower sequent of an inference rule. In case A is needed as the immediate ancestor of a principal formula in a later inference step, we simply add A by weakening, immediately before the corresponding inference.

A sequent in a pruned derivation has at most $3k+1$ formula occurrences, where k is the number of different formulas that occur in it. Therefore, there are at most $(m+1)^{3m+1}$ different sequents in the pruned cut-free derivation π_0 of $\Gamma \vdash \Delta$, if m is the number of different subformulas occurring in π_0'. Clearly, $(m+1)^{3m+1}$ also limits the height $h(\pi_0)$ of π_0. To complete the proof of Theorem 4, it remains to check that m is elementarily bounded in terms of $\bar{h}(\pi_0')$ and $h(\pi)$. To this aim it suffices to observe that every formula occurring in π_0' that does not appear in $\Gamma \vdash \Delta$ must appear in the ancestor tree of the principal formula of an application of a quantifier rule in π_0'. Therefore there are less than $2^{\bar{h}(\pi_0')}$ such formulas. On the other hand, every formula that occurs in $\Gamma \vdash \Delta$, must either occur in an axiom or as the principal formula of an inference in π, which bounds the number of such formulas by $2^{h(\pi)}$. ◀

6 Elementary bounds on the size of cut-free derivations

Let us bring together the results of the previous sections and see how bounds on the height of cut-free derivations translate into bounds on the size (number of symbols) of derivations. This yields the main result of this paper, which can be formulated as the following corollary to Theorems 2, 3, and 4.

▶ **Corollary 5.** *Let π be an $\mathbf{LI}_m^{\neg\vee}$-derivation of $\Gamma \vdash \Delta$, where each cut-formula is a prenex formula. Then there exists a cut-free $\mathbf{LI}_m^{\neg\vee}$-derivation π_0 of $\Gamma \vdash \Delta$, such that $|\pi_0| \leq f(|\pi|)$ for some elementary function f.*

Proof. The following three observations suffice.

(1) The class of elementary functions is closed under composition. Therefore, by applying first Theorem 3 (which in turn uses Theorem 2) and then Theorem 4, we obtain an elementary upper bound on the height $h(\pi_0)$ of the cut-free derivation π_0 in terms of the height $h(\pi)$ and consequently also the size $|\pi|$ of the original derivation π.

(2) As we have seen at the end of the proof of Theorem 4, π_0 can be assumed to be pruned. But both, the number of sequents as well as the number of formula occurrences in any pruned derivation are elementary bounded by its height.

(3) If the language does not contain function symbols then theorem follows immediately from (1) and (2), since then the size of (number of symbols in) a derivation is linear in the number of formula occurrences in it.

The case for languages with function symbols is somewhat more involved. We argue that every cut-free proof π_0 can be transformed into one where the size of terms occurring in it is elementary bounded by the size of terms occurring in the end-sequent of π_0 and the number of formula occurrences in π_0. To this aim, the derivation is processed upwards (i.e., from the end-sequent towards the axioms) as follows. Whenever an inference of type $(\forall, r)$ or $(\exists, l)$, introducing $\forall x A(x)$ or $\exists x A(x)$, respectively, is encountered, replace the corresponding the eigenvariable a in all ancestors of those occurrences of $\forall x A(x)$ or $\exists x A(x)$ by a new constant. Whenever an inference of type $(\forall, l)$ or $(\exists, r)$ is encountered, a term t in the upper sequents must have been replaced by a bound variable. Replace all corresponding occurrences of t in all ancestors of the principal formula of this inference by a new variable. The resulting tree of sequents γ is not a valid derivation, in general. However by applying everywhere in γ the most general simultaneous unifier of the pairs of atoms at the left and right side of leaf nodes of γ, these leaf nodes are restored to proper axioms. Finally we re-substitute fresh variables for the new constants introduced earlier. Since the applied substititution (unifier) is most general it is guaranteed that the newly introduced constants turn into variables that satisfy the eigenvariable condition. We thus obtain a proper cut-free derivation, where not only the number of formula occurrences, but also their sizes are properly bounded. (We use the fact that the size of terms in a most general unifier is exponentially bounded by size of terms in the unified atom [6].)

This concludes the proof. ◀

7 Non-elementary lower bounds for related cases of cut-elimination

To emphasize the non-triviality of our main result – elementary cut-elimination for $\mathbf{LI}_m^{\neg\vee}$ – we contrast it with the following two facts.

(1) There exists a non-elementary lower bound for cut-elimination in *classical* disjunction-free proofs with prenex atomic cuts.

(2) There exists a non-elementary lower bound for cut-elimination in intuitionistic disjunction-free proofs with *non-prenex cuts*.

To state (1) more precisely, let $\mathbf{LK}_m^{\neg\vee}$ be the classical sequent calculus with multiplicative rules, but without rules for disjunction. (In fact it does not really matter which version of the sequent calculus we use. The following result is very robust.)

We say that *elimination of certain types of cuts* for a given sequent calculus *is not elementary bounded* if there exists a sequence $\pi_1, \pi_2, \ldots$ of derivations of sizes $|\pi_1|, |\pi_2|, \ldots$, where $|\pi_i| \leq f(i)$ $(i \geq 1)$ for some elementary function f, but $|\pi_i^*| \geq g(i)$ for some non-elementary function g, where π_i^* is the shortest cut-free derivation of the end-sequent of π_i.

▶ **Theorem 6.** *The elimination of prenex atomic cuts is not elementary bounded for* $\mathbf{LK}_m^{\neg\vee}$*, even if the language does not contain function symbols.*

Proof. By Theorem 3.3 of [1] a derivation with arbitrary cut-formulas can be reduced to a derivation with only prenex cut-formulas at a quadratic expense in terms of its size. (The number of cuts may increase in this transformation.) Since we are in classical logic, disjunction-freeness is inessential: $A \vee B$ may be replaced by $\neg A \supset B$ everywhere at linear expense. We may then use the argument of Theorem 3 in Section 4 to obtain a derivation of the same end-sequent, with only prenex atomic cut-formulas. (The argument does not depend on the restricted form of intuitionistic sequents.) By Corollary 5 the increase in size is elementarily bounded. The formula sequence used by Orevkov [7] to obtain a non-elementary lower bound on cut-elimination for the classical sequent calculus **LK** does not contain function symbols. In this manner we obtain the required non-elementary lower bound for the size of cut-free derivations with respect to the size of corresponding derivations with only prenex atomic cut-formulas. ◀

Corresponding to statement (2), above, we have the following.

▶ **Theorem 7.** *The elimination of non-prenex cuts is not elementary bounded for* $\mathbf{LI}_m^{\neg\vee}$*, even if the language does not contain function symbols.*

Proof. As in the proof of Theorem 6, we may assume without loss of generality that a classical derivation does not contain disjunctions. We translate any such $\mathbf{LK}_m^{\neg\vee}$-derivation into an $\mathbf{LI}_m^{\neg\vee}$-derivation using the following inductively defined formula mapping: $A^+ = \neg\neg A$ if A is an atom, $(A \circ B)^+ = \neg\neg A^+ \circ \neg\neg B^+$ for $\circ \in \{\neg, \wedge, \supset\}$, and $(\mathsf{Q}xA)^+ = \neg\neg \mathsf{Q}xA^+$ for $\mathsf{Q} \in \{\exists, \forall\}$. Moreover let $(\neg\neg A)^-$ be $\neg A$ and write A^- instead of $(A^+)^-$. For a multiset of formulas $\Gamma = \{A_1, \ldots, A_n\}$, we define $\Gamma^+ = \{A_1{}^+, \ldots, A_n{}^+\}$ and $\Gamma^- = \{A_1{}^-, \ldots, A_n{}^-\}$. We translate a given $\mathbf{LK}_m^{\neg\vee}$-derivation of the end-sequent $A_1, \ldots, A_n \vdash B_1, \ldots, B_m$ into an $\mathbf{LI}_m^{\neg\vee}$-derivation of $A_1{}^+, \ldots, A_n{}^+, B_1{}^-, \ldots, B_m{}^- \vdash$ by induction on its depth.

Axioms: $A \vdash A$ is replaced by a derivation of $\neg\neg A, \neg A \vdash$.

Structural rules: For applications of (cut)

$$\frac{\Pi \vdash \Psi, A \quad A, \Gamma \vdash \Delta}{\Pi, \Gamma \vdash \Psi, \Delta}\,(cut) \quad \text{translates into} \quad \frac{\dfrac{\Pi^+, \Psi^-, A^- \vdash}{\Pi^+, \Psi^- \vdash A^+}\,(\neg, r) \qquad A^+, \Gamma^+, \Delta^+ \vdash}{\Pi^+, \Gamma^+, \Psi^-, \Delta^- \vdash}\,(cut)$$

The translations for weakenings and contractions are obvious.

Logical rules: We present the translation for $(\wedge_1, l)$ and $(\wedge, r)$; the other cases are analogous.

$$\frac{A, \Gamma \vdash \Delta}{A \wedge B, \Gamma \vdash \Delta}\,(\wedge_1, l) \quad \text{translates into} \quad \frac{\dfrac{\dfrac{A^+, \Gamma^+, \Delta^- \vdash}{A^+ \wedge B^+, \Gamma^+, \Delta^- \vdash}\,(\wedge_1, l)}{\Gamma^+, \Delta^- \vdash \neg(A^+ \wedge B^+)}\,(\neg, r)}{\neg\neg(A^+ \wedge B^+)[= (A \wedge B)^+], \Gamma^+, \Delta^- \vdash}\,(\neg, l)$$

$$\frac{\Gamma \vdash \Psi, A \quad \Pi \vdash \Delta, B}{\Gamma, \Pi \vdash \Psi, \Delta, A \wedge B}\,(\wedge, r) \quad \text{translates into} \quad \frac{\dfrac{\dfrac{\Gamma^+, \Psi^-, A^- \vdash}{\Gamma^+, \Psi^- \vdash A^+}\,(\neg, r) \qquad \dfrac{\Pi^+, \Delta^-, B^- \vdash}{\Pi^+, \Delta^- \vdash B^+}\,(\neg, r)}{\Gamma^+, \Pi^+, \Psi^-, \Delta^- \vdash A^+ \wedge B^+}\,(\wedge, r)}{\neg(A^+ \wedge B^+)[= (A \wedge B)^-], \Gamma^+, \Pi^+, \Psi^-, \Delta^- \vdash}\,(\neg, l)$$

Any sequence of short $\mathbf{LK}_m^{\neg\vee}$-derivations, where the corresponding shortest cut-free derivations grow non-elementarily, leads to a sequence of short $\mathbf{LI}_m^{\neg\vee}$-derivations under this translation. (See [8] for an example with weak prenex quantifiers only in the end-sequent, where disjunctions are readily removed). Note that a lower bound for shortest cut-free derivations corresponds, modulo an exponential increase, to the Herbrand complexity, i.e. the size of the shortest Herbrand sequent (see [2]). Since Herbrand sequents are propositionally valid, they remain valid when double negations are removed. ◀

▶ Remark. Since classical derivations with arbitrary many cuts can be elementarily transformed into derivations with a single cut (see [2]) we could in fact sharpen Theorem 7 accordingly.

8 Conclusion

We have shown that cut-elimination is elementary for disjunction-free intuitionistic logic with prenex cut-formulas. To achieve this result we had to come up with novel techniques. The first step – elimination of prenex atomic cuts – is specific to the case at hand: it only works for disjunction-free intuitionistic logic. The second step – trading complex prenex cut-formulas for atomic prenex and propostional cuts – uses a scheme that can also be applied in other contexts. The final step – elementary elimination of propositional cuts – although presented in a way tailored to $\mathbf{LI}_m^{\neg\vee}$, should also be adaptable to other sequent calculi, thus rendering our results of potential significance beyond disjunction-free intuitionistic logic.

References

1 Matthias Baaz and Alexander Leitsch. Cut normal forms and proof complexity. *Annals of Pure and Applied Logic*, 97(1):127–177, 1999.

2 Matthias Baaz and Alexander Leitsch. *Methods of Cut-elimination*. Springer, 2011.

3 Samuel R. Buss. An introduction to proof theory. In *Handbook of Proof Theory*, volume 137 of *Studies in Logic and the Foundations of Mathematics*, pages 1–78. Elsevier, 1998.

4 Gerhard Gentzen. Untersuchungen über das logische Schließen. I & II. *Mathematische Zeitschrift*, 39(1):176–210, 405–431, 1935.

5 Jörg Hudelmaier. Bounds for cut elimination in intuitionistic propositional logic. *Archive for Mathematical Logic*, 31(5):331–353, 1992.

6 Alberto Martelli and Ugo Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(2):258–282, 1982.

7 V.P. Orevkov. Lower bounds for increasing complexity of derivations after cut elimination. *Journal of Soviet Mathematics*, 20:2337–2350, 1982.

8 Richard Statman. Lower bounds on Herbrand's theorem. *Proc. of the Amer. Math. Soc.*, 75:104–107, 1979.

Tree Grammars for the Elimination of Non-prenex Cuts*

Stefan Hetzl and Sebastian Zivota

Institute of Discrete Mathematics and Geometry
Vienna University of Technology
Wiedner Hauptstraße 8-10, 1040 Vienna, Austria

Abstract

Recently a new connection between proof theory and formal language theory was introduced. It was shown that the operation of cut elimination for proofs with prenex Π_1-cuts in classical first-order logic corresponds to computing the language of a particular type of tree grammars. The present paper extends this connection to arbitrary (i.e. non-prenex) cuts without quantifier alternations. The key to treating non-prenex cuts lies in using a new class of tree grammars, constraint grammars, which describe the relationship of the applicability of its productions by a propositional formula.

1998 ACM Subject Classification F.4.1 Mathematical Logic, F.4.2 Grammars and Other Rewriting Systems

Keywords and phrases proof theory, cut-elimination, Herbrand's theorem, tree grammars

Digital Object Identifier 10.4230/LIPIcs.CSL.2015.110

1 Introduction

The constructive content of proofs has always been a central topic of proof theory. A helpful perspective on the constructive content of proofs in classical first-order logic is provided by Herbrand's theorem [7] (see also [2]). It states that from a valid first order formula one can obtain a quantifier-free tautology by expanding existential quantifiers to finite disjunctions of instances and universal quantifiers to finite conjunctions of instances. Provided one is willing to speak about provability instead of validity this result even extends to higher-order logic, see e.g. [17].

It is straightforward to read off a Herbrand expansion from a cut-free proof. On the other hand, proofs with cut can be non-elementarily shorter than the shortest Herbrand expansion [20, 18, 19]. Therefore, in order to compute a Herbrand expansion from a proof with cut, cut-elimination (or another equivalent normalization process) is necessary.

This paper is part of a line of research which was started in [8] and is dedicated to applying methods from formal language theory in proof theory. In [8] a class of tree grammars has been introduced which describe the Herbrand expansion obtained from proofs with prenex Π_1-cuts. The size of the grammar is bound by the size of the proof from which it is read off. The language of the grammar is a Herbrand expansion of size exponential in the size of the grammar. Thus by computing the language of this grammar, the cumbersome computational process of cut-elimination can be circumvented. These grammars owe their simplicity to the fact that they fully abstract from the propositional structure of the proof by speaking

* Supported by the Vienna Science and Technology Fund (WWTF) project VRG12-04 and by the Austrian Science Fund (FWF) project P25160.

24th EACSL Annual Conference on Computer Science Logic (CSL 2015).
Editor: Stephan Kreutzer; pp. 110–127

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

only about witness terms. There are other formalisms which allow to compute a Herbrand expansion in a way that abstracts from the propositional structure: the historically first such formalism is Hilbert's ε-calculus [14]. In [5] Gerhardy and Kohlenbach adapt Shoenfield's variant of Gödel's Dialectica interpretation to a system of pure predicate logic. Recent work, more similar to proof nets, is that of Heijltjes [6] and McKinley [16]. An approach similar to [6, 16] in the formalism of expansion trees [17] can be found in [13].

What sets the grammars introduced in [8] and treated in the present paper apart from the above-mentioned formalisms is that they do not only allow to compute a Herbrand expansion but provide a (well-understood) abstract description of its structure. On the one hand this has the consequence that problems from formal language theory such as membership, inclusion, etc. assume a proof-theoretic meaning and hence standard algorithms can be used for solving the corresponding proof-theoretic problems, usually with smaller asymptotic complexity than the naive algorithms which rely on computing the normal form(s), see e.g. [15]. On the other hand, the strong grip on the structure of a Herbrand expansion afforded by a formal grammar opens the door to the following interesting theoretical and applied investigations:

From strengthening the result of [8] one can show that all (infinitely many) normal forms of the non-erasing Gentzen reduction lead to the same Herbrand expansion, see [12]. This property has been called Herbrand-confluence in [12]. Grammars have been used for a cut-introduction algorithm in [11, 10]. This algorithm has been implemented and empirically evaluated with good results in [9] and it has recently been extended to induction in [4]. In [3] an incompressible sequence of word languages is constructed which via the result of [8] yields a sequence of first-order formulas all of whose cut-free proofs are essentially incompressible by Π_1-cuts.

All of the results so far are limited to prenex Π_1 cuts (with the exception of [1] which treats prenex Π_2 cuts) and consequently all of the applications mentioned above are so as well. In this paper, which is a generalization and an improved presentation of the results obtained in [21], we remove the limitation to prenex formulas by employing a more general class of grammars. This opens the way for extending the results and techniques of [12, 11, 10, 9, 4, 3, 1] to non-prenex cuts and induction formulas.

2 Previous Work

In this paper we will use the proof system **LK** which was introduced by Gentzen in the 1930s. It is a sequent calculus, which means that unlike most other proof systems, derivations do not operate directly on formulas, but rather on so-called sequents. A sequent is a structure of the form $\Gamma \vdash \Delta$, where Γ and Δ are multisets of formulas, respectively called the antecedent and succedent of the sequent. The natural interpretation of $\Gamma \vdash \Delta$ is "the conjunction over Γ implies the disjunction over Δ".

We shall now define the inference rules of **LK** as used in this paper. They are easily seen to be sound given the above interpretation.

▶ **Definition 1** (Rules of **LK**).

1. Axioms: $\frac{}{A \vdash A}$ with A atomic.
2. Contraction:

$$\frac{A, A, \Gamma \vdash \Delta}{A, \Gamma \vdash \Delta}\ c_l \qquad \frac{\Gamma \vdash \Delta, A, A}{\Gamma \vdash \Delta, A}\ c_r$$

3. Weakening:

$$\frac{\Gamma \vdash \Delta}{A, \Gamma \vdash \Delta}\ w_l \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, A}\ w_r$$

4. Propositional rules:

$$\frac{A, \Gamma \vdash \Delta \quad B, \Pi \vdash \Lambda}{A \vee B, \Gamma, \Pi \vdash \Delta, \Lambda}\ \vee_l \qquad \frac{\Gamma \vdash \Delta, A, B}{\Gamma \vdash \Delta, A \vee B}\ \vee_r$$

$$\frac{A, B, \Gamma \vdash \Delta}{A \wedge B, \Gamma \vdash \Delta}\ \wedge_l \qquad \frac{\Gamma \vdash \Delta, A \quad \Pi \vdash \Lambda, B}{\Gamma, \Pi \vdash \Delta, \Lambda, A \wedge B}\ \wedge_r$$

$$\frac{\Gamma \vdash \Delta, A}{\neg A, \Gamma \vdash \Delta}\ \neg_l \qquad \frac{A, \Gamma \vdash \Delta}{\Gamma \vdash \Delta, \neg A}\ \neg_r$$

5. Quantifier rules:

$$\frac{A[x \setminus t], \Gamma \vdash \Delta}{\forall x A, \Gamma \vdash \Delta}\ \forall_l \qquad \frac{\Gamma \vdash \Delta, A[x \setminus \alpha]}{\Gamma \vdash \Delta, \forall x A}\ \forall_r$$

$$\frac{A[x \setminus \alpha], \Gamma \vdash \Delta}{\exists x A, \Gamma \vdash \Delta}\ \exists_l \qquad \frac{\Gamma \vdash \Delta, A[x \setminus t]}{\Gamma \vdash \Delta, \forall x A}\ \exists_r$$

   Here, t is any term, while α is a variable that does not occur in Γ, Δ or A, called an *eigenvariable*. The inferences that use eigenvariables are called *strong quantifier inferences*, the others *weak quantifier inferences*.

6. The cut rule:

$$\frac{\Gamma \vdash \Delta, A \quad A, \Pi \vdash \Lambda}{\Gamma, \Pi \vdash \Delta, \Lambda}\ cut$$

   The formula A is called the *cut formula* of the inference. We call a cut quantified if its cut formula contains quantifiers. In the sequel, we will refer to the set of quantified cuts in a proof π as $\mathrm{QCuts}(\pi)$.

In all of these cases, the sequents above the line are called premises and the one below is called the conclusion. Additionally, the emphasized formulas in the premises are called auxiliary formulas, while the emphasized formula in the conclusion is called the main or principal formula. Note that some rules (e.g. weakening) do not have auxiliary formulas and the cut rule does not have a main formula.

Often a given formula will occur several times in a proof and these occurrences have different properties. We will mark important formula occurrences like this: $A_{[\mu]}, B_{[\nu]}$, etc.

We can formalize the notion of a formula occurrence being an ancestor of another: μ is an immediate ancestor of ν if there is an inference such that μ is its auxiliary formula and ν its main formula. The "ancestor" relation is then simply the transitive closure of the "immediate ancestor" relation. When we say that a formula is an "ancestor of the end sequent", we mean "ancestor of a formula in the end sequent".

We often visualize proofs as two-dimensional structures with axioms at the top and the conclusion at the bottom. In this context, it makes sense to say that an inference is "above" or "below" another or to talk about "left" and "right" subproofs. We also generally regard proofs as being constructed top-down, so we say for instance that the weakening rule "introduces" a formula.

Gentzen proved that every **LK**-proof can be algorithmically transformed into a cut-free proof, i.e. one that does not contain any cut inferences. The standard proof of cut-elimination in **LK** employs the following set of cut-reduction rules.

▶ **Definition 2** (Cut reduction). Let c be a cut in a proof π and let A_c be the cut formula of c. We define the following steps of cut reduction according to the inferences immediately above the cut:

1. On one side of c, there is a unary or binary inference r whose active formula is not A_c:

$$\dfrac{\dfrac{(\psi_1)}{\Gamma \vdash \Delta, A_c} \quad \dfrac{(\psi_2)}{A_c, \Pi' \vdash \Lambda'}\, r}{\Gamma, \Pi \vdash \Delta, \Lambda}\, cut \rightsquigarrow \dfrac{\dfrac{(\psi_1)}{\Gamma \vdash \Delta, A_c} \quad \dfrac{(\psi_2)}{A_c, \Pi' \vdash \Lambda'}\, cut}{\dfrac{\Gamma, \Pi' \vdash \Delta, \Lambda'}{\Gamma, \Pi \vdash \Delta, \Lambda}\, r}$$

$$\dfrac{\dfrac{(\psi_1)}{\Gamma \vdash \Delta, A_c} \quad \dfrac{(\psi_2) \quad (\psi_2)}{A_c, \Pi_1 \vdash \Lambda_1 \quad \Pi_2 \vdash \Lambda_2}\, r}{\Gamma, \Pi \vdash \Delta, \Lambda}\, cut \rightsquigarrow \dfrac{\dfrac{(\psi_1)}{\Gamma \vdash \Delta, A_c} \quad \dfrac{(\psi_2)}{A_c, \Pi_1 \vdash \Lambda_1}\, cut \quad \dfrac{(\psi_3)}{\Pi_2 \vdash \Lambda_2}}{\Gamma, \Pi \vdash \Delta, \Lambda}\, r$$

   The case where r is on the left side of c works entirely symmetrically.

2. A_c is introduced by an axiom on one side of c:

$$\dfrac{A_c \vdash A_c \quad \dfrac{(\psi)}{A_c, \Gamma \vdash \Delta}}{A_c, \Gamma \vdash \Delta}\, cut \rightsquigarrow \dfrac{(\psi)}{A_c, \Gamma \vdash \Delta}$$

3. A_c is introduced by a weakening on one side of c:

$$\dfrac{\dfrac{(\psi_1)}{\Gamma \vdash \Delta}}{\Gamma \vdash \Delta, A_c}\, w_r \quad \dfrac{(\psi_2)}{A_c, \Pi \vdash \Lambda}\, cut \rightsquigarrow \dfrac{\dfrac{(\psi_1)}{\Gamma \vdash \Delta}}{\Gamma, \Pi \vdash \Delta, \Lambda}\, w*$$

   The case where the weakening is on the right side is symmetrical.

4. A_c is the main formula of a contraction on one side of c:

$$\dfrac{\dfrac{\dfrac{(\psi_1)}{\Gamma \vdash \Delta, A_c, A_c}}{\Gamma \vdash \Delta, A_c}\, c_r \quad \dfrac{(\psi_2)}{A_c, \Pi \vdash \Lambda}\, cut}{\Gamma, \Pi \vdash \Delta, \Lambda} \rightsquigarrow \dfrac{\dfrac{\dfrac{(\psi_1)}{\Gamma \vdash \Delta, A_c, A_c} \quad \dfrac{(\psi_2'')}{A_c, \Pi \vdash \Lambda}\, cut}{\Gamma, \Pi \vdash \Delta, \Lambda, A_c} \quad \dfrac{(\psi_2')}{A_c, \Pi \vdash \Lambda}\, cut}{\dfrac{\Gamma, \Pi, \Pi \vdash \Delta, \Lambda, \Lambda}{\Gamma, \Pi \vdash \Delta, \Lambda}\, c*}$$

   Here, ψ_2' and ψ_2'' each arise from ψ_2 by replacing all eigenvariables introduced in ψ_2 with fresh copies. The case where the contraction is on the right is treated analogously.

5. $A_c = \exists x B$ and A_c is introduced by $\exists$-inferences immediately above the cut:

$$\dfrac{\dfrac{\dfrac{(\psi_1)}{\Gamma \vdash \Delta, B[x \setminus t]}}{\Gamma \vdash \Delta, \exists x B}\, \exists_r \quad \dfrac{\dfrac{(\psi_2)}{B[x \setminus t], \Pi \vdash \Lambda}}{\exists x B, \Pi \vdash \Lambda}\, \exists_l}{\Gamma, \Pi \vdash \Delta, \Lambda}\, cut \rightsquigarrow \dfrac{\dfrac{(\psi_1)}{\Gamma \vdash \Delta, B[x \setminus t]} \quad \dfrac{(\psi_2[\alpha \setminus t])}{B[x \setminus t], \Pi \vdash \Lambda}}{\Gamma, \Pi \vdash \Delta, \Lambda}\, cut$$

6. $A_c = \forall x B$: Analogous to the previous case, but with switched sides.

7. $A_c = B \wedge C$ and A_c is introduced by $\wedge$-inferences immediately above the cut:

$$\dfrac{\dfrac{\dfrac{(\psi_1)}{\Gamma_1 \vdash \Delta_1, B} \quad \dfrac{(\psi_2)}{\Gamma_2 \vdash \Delta_2, C}}{\Gamma \vdash \Delta, B \wedge C}\wedge_r \quad \dfrac{\dfrac{(\psi_3)}{B, C, \Pi \vdash \Lambda}}{B \wedge C, \Pi \vdash \Lambda}\wedge_l}{\Gamma, \Pi \vdash \Delta, \Lambda}\, cut \rightsquigarrow$$

$$\rightsquigarrow \dfrac{\dfrac{(\psi_1)}{\Gamma_1 \vdash \Delta_1, B} \quad \dfrac{\dfrac{(\psi_2)}{\Gamma_2 \vdash \Delta_2, C} \quad \dfrac{(\psi_3)}{C, B, \Pi \vdash \Lambda}}{B, \Gamma_2, \Pi \vdash \Delta_2, \Lambda}\, cut}{\Gamma, \Pi \vdash \Delta, \Lambda}\, cut$$

8. $A_c = B \vee C$: Analogous to the previous case.
9. $A_c = \neg B$ and both $\neg$-inferences introducing A_c are immediately above the cut:

$$\dfrac{\dfrac{\dfrac{(\psi_1)}{B, \Gamma \vdash \Delta}}{\Gamma \vdash \Delta, \neg B}\neg_r \quad \dfrac{\dfrac{(\psi_2)}{\Pi \vdash \Lambda, B}}{\neg B, \Pi \vdash \Lambda}\neg_l}{\Gamma, \Pi \vdash \Delta, \Lambda}\, cut \rightsquigarrow \dfrac{\dfrac{(\psi_2)}{\Pi \vdash \Lambda, B} \quad \dfrac{(\psi_1)}{B, \Gamma \vdash \Delta}}{\Gamma, \Pi \vdash \Delta, \Lambda}\, cut$$

If π' arises from π by finitely many applications of these rules, then we write $\pi \rightsquigarrow^* \pi'$.

It will often be useful to consider signed formulas, i.e. formulas annotated as either occurring in the antecedent or the consequent of a sequent. The former will be written as $A \vdash$, the latter as $\vdash A$.

▶ **Definition 3** (Herbrand set)**.** Let $\mathcal{S} = A_1, \ldots, A_m \vdash B_1, \ldots, B_n$ be a sequent. An *Herbrand set* of $\mathcal{S}$ is a set $\mathcal{H}$ for which the following two conditions hold:

1. $\mathcal{H} = \mathcal{H}^a \dot{\cup} \mathcal{H}^s$ where
   - Every element of $\mathcal{H}^a$ is of the form $A \vdash$ with A an instance of some A_i
   - Every element of $\mathcal{H}^s$ is of the form $\vdash B$ with B an instance of some B_j
2. Let $\mathcal{H}'$ be the image of $\mathcal{H}$ under the function $\begin{cases} A \vdash & \mapsto \neg A, \\ \vdash B & \mapsto B \end{cases}$. Then $\bigvee \mathcal{H}'$ is a tautology.

For the sake of simplicity, we abbreviate the latter condition as "$\mathcal{H}$ is a tautology".

It is straightforward to extract an Herbrand set of $\mathcal{S}$ from a cut-free proof of $\mathcal{S}$. By extension, one could in principle also extract an Herbrand set from a proof with cuts by performing cut elimination. There is another possibility, however: an Herbrand set can be viewed as a finite tree language by viewing both $\cdot \vdash$ and $\vdash \cdot$ as well as the propositional connectives and the predicate symbols as function symbols. Finite tree languages can be compactly represented by tree grammars. Our aim is to extract a tree grammar from a proof with cuts such that computing the language's grammar corresponds to performing cut reduction on the proof.

It is well-known that cut-elimination leads to a non-elementary increase in proof length [20, 18, 19]. On the other hand, the size of a Herbrand set is closely related to the length of a cut-free proof. Consequently, for this approach to make sense, the tree grammar must be polynomial in the length of the proof with cut. It thus provides a representation of a Herbrand set as compressed as the proof with cut.

▶ **Definition 4** (Regular tree grammar)**.** A *regular tree grammar* is a tuple $G = \langle \varphi, N, \Sigma, P \rangle$, where

1. Σ is a finite ranked alphabet; its elements are called *terminal symbols* (or *terminals* for short);

2. N is a finite set, disjoint from Σ; its elements are called *nonterminals*;
3. $\varphi \in N$ is the *starting symbol*;
4. P is the set of *productions*, i.e. elements of the form $\alpha \to t$ where $\alpha \in N$ and t is a term over $N \cup \Sigma$.

Let G be a regular tree grammar. A *derivation* in G is a finite sequence $d = \langle \varphi = t_0, t_1, \ldots, t_n \rangle$ such that t_i can be obtained from t_{i-1} by application of a production of G; that is, there is a production $\alpha_i \to s_i \in P$ such that replacing one occurrence of α_i in t_{i-1} with s_i yields t_i. We say that t_n can be derived in G.

Now we can define the language $L(G)$ of G: $L(G)$ is the set of terms over Σ that can be derived in G.

▶ **Example 5.** Consider the regular tree grammar $G = \langle \varphi, N, \Sigma, P \rangle$ with

$$N = \{\varphi, x, y\}$$
$$\Sigma = \{a/0, b/0, g/1, f/2\}$$
$$P = \{\varphi \to f(x, y), x \to a | g(y), y \to a | b\}.$$

The language of G is $\{f(a,a), f(a,b), f(g(a),a), f(g(a),b), f(g(b),a), f(g(b),b)\}$.

▶ **Definition 6.** A totally rigid tree grammar is a regular tree grammar $G = \langle \varphi, N, \Sigma, P \rangle$ with an additional restriction on derivations. Let d be a derivation of G in the sense of regular tree grammars. Then d is a derivation of the totally rigid grammar G if for each $\alpha \in N$, at most one production beginning with α is used in d.

▶ **Example 7.** Let G be the grammar from Example 5. If we view it as a totally rigid grammar, its language is reduced to $\{f(a,a), f(a,b), f(g(a),a), f(g(b),b)\}$.

It is easy to see that the language of a totally rigid grammar is always finite.

The following theorem was proved in [8].

▶ **Theorem 8.** *Let π be a proof with the following properties:*
1. *The end sequent of π is of the form $\vdash \exists \bar{x} A$ with A quantifier-free;*
2. *All cut formulas in π are of the form $\exists y B$ with B quantifier-free.*

Then there is a totally rigid tree grammar $G(\pi)$ such that $L(G(\pi))$ is an Herbrand set of $\exists \bar{x} A$. Moreover, if $|G|$ is the number of productions of G and $|\pi|$ the number of inferences in π, then $|G| \leq |\pi|$.

In this paper we will generalize this result to non-prenex cut formulas and arbitrary end sequents. In order to do that, we will need more powerful grammars.

3 Constraint grammars

Totally rigid grammars are obtained from regular tree grammars by placing restrictions on how many productions can be used per nonterminal in a derivation. Similarly, constraint grammars allow us to restrict which combinations of productions of different nonterminals can be used. This is essential if we want to deal with cut formulas that are non-prenex and contain more than one quantifier.

▶ **Definition 9** (Constraint grammar)**.** A constraint grammar is a tuple $G = \langle \varphi, N, \Sigma, P, \mathcal{C} \rangle$ consisting of a totally rigid grammar $G' = \langle \varphi, N, \Sigma, P \rangle$ together with a *constraint formula* $\mathcal{C}$, which is a propositional formula that uses the productions in P as atoms.

When writing constraint formulas, we will use the symbol "$\to$" to denote productions and "$\Rightarrow$" for implications.

Any derivation d of the underlying totally rigid grammar of G induces an interpretation v_d of $\mathcal{C}$ in the following manner: If $\alpha \in N$ such that α does not occur in d, then $v_d(p) = \top$ for all $p \in P_\alpha$. If α occurs in d, v_d evaluates the α-productions used in d as $\top$ and the others as $\bot$. This leads to the definition of a valid derivation of G: d is valid iff $v_d(\mathcal{C}) = \top$ (i.e. v_d is a model of $\mathcal{C}$).

A term over Σ is derivable in G if it is derivable in G' via a valid derivation.

Note that determining whether a given derivation is valid for G can be done in linear time relative to the size of d and $\mathcal{C}$.

▶ **Example 10.** Let G be the totally rigid grammar from Example 7. If we extend it to a constraint grammar G' by adding the constraint formula $\mathcal{C} := x \to a \vee y \to a$, then $L(G') = \{f(a,a), f(a,b), f(g(a),a)\}$.

4 The grammar of a proof

In this section we will give the central definition of this paper: the constraint grammar induced by a proof.

When working in sequent calculus, it is customary to distinguish between weak and strong quantifiers. Briefly, a quantifier is said to be "strong" if it is universal and below an even number of negations or existential and below an odd number of negations. Conversely, it is called "weak" if it is universal and below an odd number of negations or existential and below an even number. Note that in this context, both the left side of an implication and the antecedent of a sequent count as one negation each.

In the sequel, we always place some restrictions on the proofs we consider.

- The names of bound variables in the end sequent are distinct. This can always be ensured via renaming.
- There are no strong quantifiers in the end sequent. This assumption is justified because we can perform validity-preserving Skolemization, i.e. replace all strong quantifiers by Skolem symbols.
- Each cut formula contains only weak or strong quantifiers, but not both. We call a cut formula Σ_1 or Π_1 accordingly.

From now on, we will call proofs with these properties *simple*.

The above restriction on cut formulas allows us to define the "weak side" and the "strong side" of a cut: Let A_c be the cut formula of a cut c and assume that A_c contains quantifiers. If A_c is Σ_1, each quantifier in c is introduced via a weak quantifier inference in the left subproof of c and a strong inference in the right subproof. Consequently, the left and right subproof are called the "weak" and "strong" side, respectively. In the case of a Π_1 cut formula, the sides are switched. Each quantifier in A_c may be introduced several times on both the weak and the strong side of c; this happens via eigenvariables on the the strong side and arbitrary terms on the weak side. We refer to those eigenvariables and terms as "belonging to" or "being associated with" the quantifier.

We shall now define a constraint grammar $G(\pi) = \langle \varphi, N(\pi), \Sigma, P(\pi), \mathcal{C}(\pi) \rangle$ piece by piece.

▶ **Definition 11** (Terminals and nonterminals of $G(\pi)$)**.** Let π be a simple proof of $A_1, \ldots, A_m \vdash B_1, \ldots, B_n$.

- Terminals: The terminal symbols Σ of $G(\pi)$ consist of the language of π together with a new symbol w. w will be used to mark places where a formula is introduced by weakening in π.
- Nonterminals: We define sets $\mathrm{N}_{\mathrm{ES}}(\pi)$ and $\mathrm{N}_{\mathrm{Cuts}}(\pi)$. Let $BV(A)$ be the set of bound variables in the formula A and φ a new symbol. Then $\mathrm{N}_{\mathrm{ES}}(\pi) = \{\varphi\} \cup \bigcup_{i=1}^{m} BV(A_i) \cup \bigcup_{i=1}^{n} BV(B_i)$. Since there are no strong quantifiers in the end sequent of π, all strong quantifier inferences must act on ancestors of cut formulas. Thus each eigenvariable is uniquely associated with a particular cut. We write $EV(c)$ for the eigenvariables associated with cut c and $EV(\pi)$ for all eigenvariables in π. This leads to $\mathrm{N}_{\mathrm{Cuts}}(\pi) = EV(\pi)$.

Finally, $N(\pi) := \mathrm{N}_{\mathrm{ES}}(\pi) \cup \mathrm{N}_{\mathrm{Cuts}}(\pi)$.

For any formula A, let $\widehat{A}$ be the matrix of A, i.e. the formula that results from deleting all quantifiers from A.

▶ **Definition 12** (Productions of $G(\pi)$)**.** Let π be a simple proof of $A_1, \dots, A_m \vdash B_1, \dots, B_n$. We define sets $\mathrm{P}_{\mathrm{ES}}(\pi)$ and $\mathrm{P}_{\mathrm{Cuts}}(\pi)$:

For $i = 1, \dots, m$, $(\varphi \to \widehat{A_i} \vdash) \in \mathrm{P}_{\mathrm{ES}}(\pi)$. For $j = 1, \dots, n$, $(\varphi \to \vdash \widehat{B_j}) \in \mathrm{P}_{\mathrm{ES}}(\pi)$.

For $x \in \mathrm{N}_{\mathrm{ES}}(\pi)$, if π contains an inference $\dfrac{\Gamma \vdash \Delta, A[x \setminus t]}{\Gamma \vdash \Delta, \exists x A}\ \exists_r$, then $x \to t \in \mathrm{P}_{\mathrm{ES}}(\pi)$. Moreover, if x is introduced by weakening at least once in π, then $x \to w \in \mathrm{P}_{\mathrm{ES}}(\pi)$.

Let $\alpha \in \mathrm{N}_{\mathrm{Cuts}}(\pi)$, then α is used to introduce a strong quantifier on some variable z in a cut formula. If the weak side of the cut contains an inference $\dfrac{\Gamma \vdash \Delta, B[z \setminus s]}{\Gamma \vdash \Delta, \exists z B}\ \exists_r$, then $z \to s \in \mathrm{P}_{\mathrm{Cuts}}(\pi)$. Moreover, if z is introduced by weakening at least once on the weak side of the cut, then $z \to w \in \mathrm{P}_{\mathrm{Cuts}}(\pi)$. $P(\pi) := \mathrm{P}_{\mathrm{ES}}(\pi) \cup \mathrm{P}_{\mathrm{Cuts}}(\pi)$.

▶ **Definition 13** (Constraint formula of $G(\pi)$)**.** Let π be a simple proof and μ any formula occurrence in π. We define a formula $q(\mu, \pi)$ by induction:

- If μ is quantifier-free, then $q(\mu, \pi) := \top$.
- If μ is introduced by a weakening, then let $z_1, \dots, z_k$ be the weakly bound variables in μ. There are two cases to consider. If μ is ancestor of a cut formula, then for each i let $\alpha_{i,1}, \dots \alpha_{i,n_i}$ be the eigenvariables used to introduce the quantifier over z_i on the strong side and $q(\mu, \pi) := \bigwedge_{i=1}^{k} \bigwedge_{j=1}^{n_i} \alpha_{i,j} \to w$. If μ is ancestor of a formula in the end sequent, then $q(\mu, \pi) := \bigwedge_{i=1}^{k} z_i \to w$.
- If μ is introduced by a quantifier rule, i.e. $\dfrac{\Gamma \vdash \Delta, (A[x \setminus t])_{[\mu']}}{\Gamma \vdash \Delta, (\exists x A)_{[\mu]}}\ \exists_r$, then we make a similar case distinction as in the previous case. If μ is ancestor of a cut formula, then let $\alpha_1, \dots, \alpha_n$ be the eigenvariables of the quantifier of x on the strong side of the cut and $q(\mu, \pi) := \left(\bigvee_{j=1}^{n} \alpha_j \to t\right) \wedge q(\mu', \pi)$. Otherwise, $q(\mu, \pi) := x \to t \wedge q(\mu', \pi)$. The case of a $\forall_l$-inference is analogous.
- If μ is introduced by a $\wedge_r$-rule, as in $\dfrac{\Gamma_1 \vdash \Delta_1, A_{[\nu_1]} \quad \Gamma_2 \vdash \Delta_2, B_{[\nu_2]}}{\Gamma \vdash \Delta, (A \wedge B)_{[\mu]}}\ \wedge_r$, then $q(\mu, \pi) := q(\nu_1, \pi) \wedge q(\nu_2, \pi)$. An $\vee_l$-inference is treated analogously.
- If μ is introduced by a $\wedge_l$-rule, as in $\dfrac{A_{[\nu_1]}, B_{[\nu_2]}, \Gamma \vdash \Delta}{(A \wedge B)_{[\mu]}, \Gamma \vdash \Delta}\ \wedge_l$, then $q(\mu, \pi) := q(\nu_1, \pi) \wedge q(\nu_2, \pi)$, and analogously for $\vee_r$.
- If μ arises from a contraction on the right, i.e. $\dfrac{\Gamma \vdash \Delta, A_{[\nu_1]}, A_{[\nu_2]}}{\Gamma \vdash \Delta, A_{[\mu]}}\ c_r$, then $q(\mu, \pi) := q(\nu_1, \pi) \vee q(\nu_2, \pi)$, and analogously for a contraction on the left.

- If μ is introduced by a $\neg_r$ rule, as in $\dfrac{\Gamma, A_{[\nu]} \vdash \Delta}{\Gamma \vdash \Delta, (\neg A)_{[\mu]}}\ \neg_r$, then $q(\mu, \pi) := q(\nu, \pi)$. A $\neg_l$-inference is treated analogously.
- We skip over all inferences whose active formula is not μ.

Now let A be any formula and $\mu_1, \ldots, \mu_m$ and $\nu_1, \ldots, \nu_n$ the occurrences of A in the antecedent and the succedent of the end sequent, respectively. Then

$$\mathcal{C}_A^{ant}(\pi) := (\varphi \to \widehat{A} \vdash) \Rightarrow \bigvee_{i=1}^{m} q(\mu_i, \pi)$$

$$\mathcal{C}_A^{suc}(\pi) := (\varphi \to \vdash \widehat{A}) \Rightarrow \bigvee_{j=1}^{n} q(\nu_j, \pi)$$

This yields the constraint formula of the end sequent:

$$\mathcal{C}_{\mathrm{ES}}(\pi) := \bigwedge_{A \in ES(\pi)} (\mathcal{C}_A^{ant}(\pi) \wedge \mathcal{C}_A^{suc}(\pi))$$

Furthermore, let $c \in \mathrm{QCuts}(\pi)$ and μ_0 the weak occurrence of its cut formula. Then

$$\mathcal{C}_c(\pi) := q(\mu_0, \pi).$$

Finally we obtain

$$\mathcal{C}(\pi) := \mathcal{C}_{\mathrm{ES}}(\pi) \wedge \bigwedge_{c \in \mathrm{QCuts}(\pi)} \mathcal{C}_c(\pi), \tag{1}$$

the constraint formula of π.

▶ **Definition 14** (Grammar of a proof)**.** Let π be a simple proof. The constraint grammar $G(\pi) := \langle \varphi, N(\pi), \Sigma(\pi), P(\pi), \mathcal{C}(\pi) \rangle$ is called the grammar of π.

The purpose of $\mathcal{C}(\pi)$ is to describe the set of tuples of instances that actually occur in the proof.

▶ **Example 15.** Let π be the following proof:

$$\dfrac{\dfrac{\overset{(\pi_1)}{P(f(a,c)) \vee Q(b) \vdash A_c} \quad \overset{(\pi_2)}{A_c \vdash \exists x P(x), \exists y Q(y)}\ cut_{[c]}}{P(f(a,c)) \vee Q(b) \vdash \exists x P(x), \exists y Q(y)}\ \vee_r}{P(f(a,c)) \vee Q(b) \vdash \exists x P(x) \vee \exists y Q(y)}$$

where

$$\pi_1 = \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{P(f(a,c)) \vdash P(f(a,c))}{P(f(a,c)) \vdash \exists z_2 P(f(a,z_2))}\ \exists_r}{P(f(a,c)) \vdash \exists z_2 P(f(a,z_2)), Q(a)}\ w_r}{P(f(a,c)) \vdash \exists z_2 P(f(a,z_2)) \vee Q(a)}\ \vee_r \quad \dfrac{\dfrac{\dfrac{Q(b) \vdash Q(b)}{Q(b) \vdash \exists z_2 P(f(b,z_2)), Q(b)}\ w_r}{Q(b) \vdash \exists z_2 P(f(b,z_2)) \vee Q(b)}\ \vee_r}{Q(b) \vdash A_c}\ \exists_r}{P(f(a,c)) \vdash A_c}\ \exists_r \quad \dfrac{}{Q(b) \vdash A_c}}{P(f(a,c)) \vee Q(b) \vdash A_c, A_c}\ \vee_l}{P(f(a,c)) \vee Q(b) \vdash A_c}\ c_r$$

$$\pi_2 = \dfrac{\dfrac{\dfrac{\dfrac{P(f(\alpha,\beta)) \vdash P(f(\alpha,\beta))}{P(f(\alpha,\beta)) \vdash \exists x P(x)}\ \exists_r}{\exists z_2 P(f(\alpha,z_2)) \vdash \exists x P(x)}\ \exists_l \quad \dfrac{Q(\alpha) \vdash Q(\alpha)}{Q(\alpha) \vdash \exists y Q(y)}\ \exists_r}{\exists z_2 P(f(\alpha,z_2)) \vee Q(\alpha) \vdash \exists x P(x), \exists y Q(y)}\ \vee_l}{A_c \vdash \exists x P(x), \exists y Q(y)}\ \exists_l$$

Here, A_c is the formula $\exists z_1(\exists z_2 P(f(z_1, z_2)) \vee Q(z_1))$. The various parts of $G(\pi)$ are:

- Terminals: $P/1, Q/1, f/2, a/0, b/0, c/0, w/0$
- Nonterminals: $\varphi, x, y, \alpha, \beta$
- Productions:

$$\begin{aligned}
&\varphi \to P(f(a,c)) \vee Q(b) \vdash \ \mid \ \vdash P(x) \vee Q(y),\\
&x \to f(\alpha, \beta),\\
&y \to \alpha,\\
&\alpha \to a|b,\\
&\beta \to c|w.
\end{aligned}$$

- Constraint formula:

$$\begin{aligned}
\mathcal{C}_{\mathrm{ES}}(\pi) &= ((\varphi \to \vdash P(x) \vee Q(y)) \Rightarrow (x \to f(\alpha,\beta) \wedge y \to \alpha))\\
&\quad \wedge ((\varphi \to P(f(a,c)) \vee Q(b) \vdash) \Rightarrow \top),\\
\mathcal{C}_c(\pi) &= (\alpha \to a \wedge \beta \to c) \vee (\alpha \to b \wedge \beta \to w),\\
\mathcal{C}(\pi) &= \mathcal{C}_{\mathrm{ES}}(\pi) \wedge \mathcal{C}_c(\pi).
\end{aligned}$$

Consequently, the language of $G(\pi)$ is

$$L(G(\pi)) = \{P(f(a,c)) \vee Q(b) \vdash \ , \ \vdash P(f(a,c)) \vee Q(a) \ , \ \vdash P(f(b,w)) \vee Q(b)\}$$

The following theorem is the main result of this paper.

▶ **Theorem 16.** *Let π be a simple proof of $\Gamma \vdash \Delta$. Then $L(G(\pi))$ is an Herbrand set of $\Gamma \vdash \Delta$.*

The first step towards proving this result will be to show that $L(G(\pi))$ is a Herbrand set if π is almost cut-free, more precisely: if π contains only cuts without quantifiers.

▶ **Lemma 17.** *Let π be a simple proof of $\Gamma \vdash \Delta$ in which no cut formula contains a quantifier. Then $L(G(\pi))$ is an Herbrand set of $\Gamma \vdash \Delta$.*

Proof. By induction on the length of π. The case of π a one-line proof of an axiom is trivial. Now we consider the various possibilities for the lowest inference of π. We only consider one of the cedents in each case; the other one is treated analogously. Moreover, we only show the validity of the language; the fact that it consists of instances of the end sequent is immediately obvious. Recall that we say "$L(G(\pi))$ is a tautology" to mean "the image of $L(G(\pi))$ under the function $\begin{cases} A \vdash & \mapsto \neg A, \\ \vdash B & \mapsto B \end{cases}$ is a tautology".

- Weakening: Let $\pi = \dfrac{\begin{matrix}(\pi')\\ \Gamma \vdash \Delta\end{matrix}}{\Gamma \vdash \Delta, A}\, w_r$. Let $x_1, \ldots, x_n$ be the bound variables in A. Obviously $L(G(\pi)) = L(G(\pi')) \cup \{\vdash A[x_1 \setminus w, \ldots, x_n \setminus w]\}$ is an Herbrand set if $L(G(\pi'))$ is.
- Contraction: Let $\pi = \dfrac{\begin{matrix}(\pi')\\ \Gamma \vdash \Delta, A_{[\mu_1]}, A_{[\mu_2]}\end{matrix}}{\Gamma \vdash \Delta, A_{[\mu]}}\, c_r$. It is easy to see that the constraint formulas, nonterminals and productions are unchanged between π and π'. Therefore, $L(G(\pi)) = L(G(\pi'))$.

- Negation: Let $\pi = \dfrac{\overset{(\pi')}{A_{[\mu']}, \Gamma \vdash \Delta}}{\Gamma \vdash \Delta, (\neg A)_{[\mu]}}\ \neg_r$. Observe that if

$$\mathcal{C}^{ant}_A(\pi') = (\varphi \to \widehat{A} \vdash) \Rightarrow (q(\mu', \pi') \vee \mathcal{B}_1),$$
$$\mathcal{C}^{suc}_{\neg A}(\pi') = (\varphi \to \vdash \neg\widehat{A}) \Rightarrow \mathcal{B}_2,$$

then

$$\mathcal{C}^{ant}_A(\pi) = (\varphi \to \widehat{A} \vdash) \Rightarrow \mathcal{B}_1,$$
$$\mathcal{C}^{suc}_{\neg A}(\pi) = (\varphi \to \vdash \neg\widehat{A}) \Rightarrow (q(\mu, \pi) \vee \mathcal{B}_2).$$

It follows that $\vdash \neg\widehat{A}$ is derivable in $G(\pi)$ iff $\widehat{A} \vdash$ is derivable in $G(\pi')$. Clearly, $L(G(\pi))$ is an Herbrand set if $L(G(\pi'))$ is.

- Disjunction: Let $\pi = \dfrac{\overset{(\pi')}{\Gamma \vdash \Delta, A_{[\mu_1]}, B_{[\mu_2]}}}{\Gamma \vdash \Delta, (A \vee B)_{[\mu]}}\ \vee_r$. The language of $G(\pi')$ can be written as $L_{\vdash A} \cup L_{\vdash B} \cup L_{\Gamma\vdash\Delta}$, where $L_{\vdash A}$ contains those derivable formulas that are obtained by starting with the production $\varphi \to \vdash \widehat{A}$, and analogously for $L_{\vdash B}$ and $L_{\Gamma\vdash\Delta}$. Note that these sets are not necessarily disjoint; for instance, A and B might coincide or one of them might occur in the context. In $G(\pi)$, the nonterminals and the constraint formula are unchanged, but φ has the production $\varphi \to \vdash \widehat{A} \vee \widehat{B}$. This means that $L(G(\pi)) = L_{\vdash A\vee B} \cup L_{\Gamma\vdash\Delta}$, where $L_{\vdash A\vee B} = \{\vdash A' \vee B' : \vdash A' \in L_A, \vdash B' \in L_B\}$. It follows that if $L(G(\pi'))$ is a tautology, so is $L(G(\pi))$.

- Conjunction: Let $\pi = \dfrac{\overset{(\pi')}{\Gamma \vdash \Delta, A_{[\mu_1]}} \quad \overset{(\pi'')}{\Pi \vdash \Lambda, B_{[\mu_2]}}}{\Gamma, \Pi \vdash \Delta, \Lambda, (A \wedge B)_{[\mu]}}\ \wedge_r$. Similarly to the previous case, write

$$L(G(\pi')) = L_{\vdash A} \cup L_{\Gamma\vdash\Delta},$$
$$L(G(\pi'')) = L_{\vdash B} \cup L_{\Pi\vdash\Lambda},$$
$$L(G(\pi)) = L_{\vdash A\wedge B} \cup L_{\Gamma\vdash\Pi} \cup L_{\Delta\vdash\Lambda}$$

Given any interpretation of the atoms in $L(G(\pi))$, there are two possibilities. If any element of $L_{\Gamma\vdash\Pi}$ or $L_{\Delta\vdash\Lambda}$ is true under the interpretation, we are done. If all of them are false, then some $\vdash A' \in L_{\vdash A}$ and $\vdash B' \in L_{\vdash B}$ must be true by induction. This means that $\vdash A' \wedge B' \in L_{\vdash A\wedge B}$ is also true. Thus, $L(G(\pi))$ is a tautology.

- Existential quantifier: Let $\pi = \dfrac{\overset{(\pi')}{\Gamma \vdash \Delta(A[x \setminus t])_{[\mu']}}}{\Gamma \vdash \Delta, (\exists x A)_{[\mu]}}\ \exists_{r[\iota]}$. In $G(\pi)$, the production $\varphi \to \widehat{A}[x \setminus t]$ that exists in $G(\pi')$ is replaced by $\varphi \to \widehat{A}$ and $x \to t$. If $\mathcal{C}(\pi')$ contains the subformulas

$$\mathcal{C}^{suc}_{A[x \setminus t]}(\pi') = (\varphi \to \widehat{A}[x \setminus t]) \Rightarrow (q(\mu', \pi') \vee \mathcal{B}_1),$$
$$\mathcal{C}^{suc}_{\exists x A}(\pi') = (\varphi \to \widehat{A}) \Rightarrow \mathcal{B}_2,$$

then $\mathcal{C}(\pi)$ contains

$$\mathcal{C}^{suc}_{A[x \setminus t]}(\pi) = (\varphi \to \widehat{A}[x \setminus t]) \Rightarrow \mathcal{B}_1,$$
$$\mathcal{C}^{suc}_{\exists x A}(\pi) = (\varphi \to \widehat{A}) \Rightarrow (((x \to t) \wedge q(\mu', \pi)) \vee \mathcal{B}_2).$$

If $\vdash C$ (or $C \vdash$) $\in L(G(\pi'))$ is an instance of a formula in the context, then it can still be derived in $G(\pi)$. If $\vdash C$ is an instance of $A[x \setminus t]$ and d' a derivation leading to it, d' must begin with $\varphi \to \widehat{A}[x \setminus t]$. We can transform d' into a valid derivation d of $G(\pi)$ by replacing this first step with $\varphi \to \widehat{A} \to \widehat{A}[x \setminus t]$. Thus, the two languages coincide.

- Cut: Let

$$L(G(\pi')) = L'_{\vdash A} \cup L_{\Gamma \vdash \Delta},$$
$$L(G(\pi'')) = L''_{A \vdash} \cup L_{\Pi \vdash \Lambda},$$
$$L(G(\pi)) = L_{\Gamma,\Pi \vdash \Delta,\Lambda} = L_{\Gamma,\Pi \vdash} \cup L_{\vdash \Delta,\Lambda}$$

as before. Our goal is to show that $\bigvee L_{\Gamma,\Pi \vdash \Delta,\Lambda}$ is tautological. First of all, note that the cut formula is quantifier-free and hence its occurrences only contribute one instance each to the languages of their respective grammars, namely respectively $\vdash A$ and $A \vdash$. Now pick any interpretation. If any element of $L_{\Gamma,\Pi \vdash}$ is true, we are done. Otherwise either an element of $L_{\vdash \Delta}$ or A must be true because $L(G(\pi'))$ is a tautology. In the former case we are, again, done; in the latter case, an element of $L(G(\pi''))$ must be true. This element can be neither A itself nor anything in $L_{\vdash \Pi}$, so it must be an element of L_Λ. Thus, under each interpretation, an element of $L(G(\pi))$ evaluates to true. ◀

5 Cut elimination and grammars

▶ **Definition 18** ($\leq$ relation for formulas)**.** We define a relation $\leq$ between formulas: $A \leq B$ if B can be obtained by replacing occurrences of w in A with terms. Note that different occurrences of w may be replaced with different terms. A strict semantic definition of $A \leq B$ can be achieved in the following manner: Let A' be the formula that results from making all the occurrences of w in A distinct, i.e. replacing each w with a new constant symbol w_i. Then $\forall \bar{w} A' \Rightarrow B$ is valid.

$\leq$ is clearly transitive and reflexive. For sets of formulas M and N, let $M \leq N$ if for each $A \in M$ there is a $B \in N$ such that $A \leq B$.

▶ **Lemma 19.** *Let π, π' be simple proofs and $\pi \rightsquigarrow \pi'$ by one of the cut reduction steps defined in 2, except contraction. Then $L(G(\pi')) \leq L(G(\pi))$.*

Proof. None of the reduction steps for rule permutations, axioms, or propositional inferences change the grammar[1]. Therefore, the only interesting cases are those of quantifier rules and weakening. Let us consider quantifier inferences first. Let ι be the quantifier inference under discussion. Obviously, there is only a single production for α in $G(\pi)$, namely $\alpha \to t$, and

$$\mathcal{C}_c(\pi) = \alpha \to t \wedge \mathcal{C}'_c(\pi).$$

In $G(\pi')$, α and its single production are deleted and any production $\beta \to s \in P(\pi)$ is replaced by $\beta \to s[\alpha \setminus t]$. Moreover, the constraint formula of $G(\pi')$ is obtained by replacing $\mathcal{C}_c(\pi)$ with $\mathcal{C}'_c(\pi)$ and α with t, respectively, in $\mathcal{C}(\pi)$. Clearly, all other cuts are unaffected by the transformation.

Let $d = \varphi \to \ldots \to C$ be a valid derivation of $G(\pi)$. The derivation d' of $G(\pi')$ that is obtained from d by deleting all applications of $\alpha \to t$ and then simply replacing α with t obviously generates C, so we only need to show that it is valid. There are two cases to

[1] Note, though, that a binary propositional reduction "moves" a conjunction from the constraint formula of one cut to between constraint formulas of two cuts, which makes no semantic difference.

consider here. If α does not occur in d, then $d = d'$ and the validity of d' follows immediately. Now suppose α occurs in d. For every atom $\beta \to s$ in $\mathcal{C}(\pi)$ such that $v_d(\beta \to s) = \top$, clearly $v_{d'}(\beta \to s[\alpha \setminus t]) = \top$. This implies $v_{d'}(\mathcal{C}(\pi')) = \top$ and hence d' is valid in $G(\pi')$. The other direction is proved similarly. Thus, $L(G(\pi')) = L(G(\pi))$.

Now let's consider the case that a cut formula is introduced by weakening on the weak side of c. Let μ be a formula occurrence in the premise on the strong side of c, but not the cut formula, and let $x_1, \ldots, x_n$ be the bound variables in μ. Assume further that μ is an ancestor of a formula occurrence ν in the succedent of the end sequent. If d is any derivation in $G(\pi)$ that begins with $\varphi \to\vdash \widehat{\nu}$, the x_i are eventually replaced with terms t_i in d. Call the end result of this derivation $A(t_1, \ldots, t_n)$. Now consider that in π', μ is introduced via weakening. This means that each x_i has the production $x_i \to w$ in $G(\pi')$. Consequently, we can construct a derivation d' of $A(w, \ldots, w)$ that is valid for $G(\pi')$. Clearly, $A(w, \ldots, w) \leq A(t_1, \ldots, t_n)$.

If the cut formula is introduced by weakening on the strong side, c has no nonterminals and hence contributes nothing to the grammar. In this case, removing the cut clearly changes nothing. ◀

We will need a minor proof transformation that allows us to make some simplifying assumptions later on. The motivation behind this transformation is the following observation: It is never necessary to introduce a strong quantifier twice on the same branch of a proof.

▶ **Definition 20** (Pruning). Let π and π' be proofs of the same end sequent. We say that π' is the result of *"pruning"* π, written as $\pi \rightsquigarrow \pi'$, if π' is obtained from π by the following subproof transformation:

$$\dfrac{\dfrac{\dfrac{\dfrac{(\psi)}{A[x \setminus \beta], \Gamma'' \vdash \Delta''}}{\exists x A, \Gamma'' \vdash \Delta''}\,\exists_l}{\vdots}}{\dfrac{\dfrac{A[x \setminus \alpha], \Gamma' \vdash \Delta'}{\exists x A, \Gamma' \vdash \Delta'}\,\exists_l}{\vdots}}{\dfrac{C[\exists x A], C[\exists x A], \Gamma \vdash \Delta}{C[\exists x A], \Gamma \vdash \Delta}\,c_l} \quad \rightsquigarrow \quad \dfrac{\dfrac{\dfrac{\dfrac{(\psi[\beta \setminus \alpha])}{A[x \setminus \alpha], \Gamma'' \vdash \Delta''}}{\exists x A, A[x \setminus \alpha], \Gamma'' \vdash \Delta''}\,w_l}{\vdots}}{\dfrac{\dfrac{A[x \setminus \alpha], A[x \setminus \alpha], \Gamma' \vdash \Delta'}{A[x \setminus \alpha], \Gamma' \vdash \Delta'}\,c_l}{\exists x A, \Gamma' \vdash \Delta'}\,\exists_l}{\vdots}}{\dfrac{C[\exists x A], C[\exists x A], \Gamma \vdash \Delta}{C[\exists x A], \Gamma \vdash \Delta}\,c_l}$$

We say that a proof is *"pruned"* if it cannot be pruned further.

▶ **Lemma 21.** *Let π, π' be simple proofs such that π' is obtained from π by pruning. Then $L(G(\pi')) \subseteq L(G(\pi))$.*

Proof. We show that every derivation that is valid for $G(\pi')$ can be transformed to one that is valid for $G(\pi)$. This is possible because the eigenvariables that are identified by pruning are associated with the same quantifier and thus have the same productions. Given a derivation d' that is valid for $G(\pi')$, suppose α, β are as in the definition of pruning and d' uses a production $\nu \to t[\beta \setminus \alpha]$. Clearly, $\nu \to t$ is a production of $G(\pi)$ and due to the above considerations, α and β have the same productions. We can therefore replace the step $\nu \to t[\beta \setminus \alpha]$ in d' with $\nu \to t$ and add the required β-productions at any point after that. ◀

For technical reasons, we only allow the reduction of minimal cuts in this lemma. We call a cut minimal if its strong side does not intersect with the weak side of any other cut. It

is easy to prove that a minimal cut always exists. The nonterminals of a minimal cut never occur on the right side of productions of other cuts.

▶ **Lemma 22.** *Let π be a pruned simple proof and c a minimal cut in π. If $\pi \rightsquigarrow \pi'$ by reducing c according to a contraction rule, then $L(G(\pi)) = L(G(\pi'))$.*

Proof. We assume that c is Σ_1; the case of a Π_1-cut can be treated by switching the strong and weak sides. Let $G(\pi') = \langle \varphi, N', \rho', \Sigma, P', \mathcal{C}' \rangle$.

First, suppose that the contraction that is reduced is on the left-hand (weak) side of c. The first thing we note is that the only nonterminals that are affected by the proof transformation are those introduced in ψ_2. Due to the minimality of c, there are no quantified cuts in ψ_2 and hence the only eigenvariables therein are those of cuts below c and those of c itself. Let $EV(c) = \{\alpha_1, \ldots, \alpha_n\}$. In $G(\pi')$, each α_i is replaced by two new copies α_i' and α_i''. Moreover, if $\tilde{c}$ is a cut in π such that c is on the strong side of $\tilde{c}$, then there might be eigenvariables of $\tilde{c}$ that are introduced within ψ_2. Let $\beta_1, \ldots, \beta_m$ be all such eigenvariables; it follows that π' contains two new copies β_i', β_i'' for each of them.

Let us now consider the effects of the reduction on the nonterminals and productions of the end sequent. Let $p : z \to t$ be a production of the end sequent. If t contains no α_i or β_i, p is unchanged; otherwise, p arises from some quantifier inference in ψ_2 that is duplicated along with ψ_2. This means that in $G(\pi')$, p is replaced by two new productions

$$p' : z \to t[\alpha_1 \setminus \alpha_1', \ldots, \alpha_n \setminus \alpha_n', \beta_1 \setminus \beta_1', \ldots, \beta_m \setminus \beta_m'],$$
$$p'' : z \to t[\alpha_1 \setminus \alpha_1'', \ldots, \alpha_n \setminus \alpha_n'', \beta_1 \setminus \beta_1'', \ldots, \beta_m \setminus \beta_m''].$$

Now we consider the rest of the grammar. If μ' and μ'' are the two occurrences of A_c on the weak side of c, then one of them is arbitrarily designated as the cut formula of c' and the other as the cut formula of c''; w.l.o.g we assume that μ' is the cut formula of c' and μ'' the cut formula of c''. The productions of α_i are split between α_i' and α_i'' accordingly, that is, if $\alpha_i \to t$ is a production of $G(\pi)$ and t introduces a quantifier in μ', then $\alpha_i' \to t$ is a production of $G(\pi')$ and analogously if t introduces a quantifier in μ''. Note that these cases are not mutually exclusive.

As for the β_i, each of them originates from a cut below c whose weak side is entirely unaffected by the duplication of ψ_2, so β_i' and β_i'' simply inherit the productions of β_i.

Let us now turn to the constraint formula. First of all, the constraint formula of c is necessarily of the form $\mathcal{B}' \vee \mathcal{B}''$; it follows that the constraint formulas of c' and c'' are $\mathcal{B}'$ and $\mathcal{B}''$, respectively, up to replacement of nonterminals by their fresh copies:

$$\mathcal{C}_{c'}(\pi') = \mathcal{B}'\{\alpha_1 \setminus \alpha_1', \ldots, \alpha_n \setminus \alpha_n'\},$$
$$\mathcal{C}_{c''}(\pi') = \mathcal{B}''\{\alpha_1 \setminus \alpha_1'', \ldots, \alpha_n \setminus \alpha_n''\}$$

Moreover, if ν is any formula occurrence in the conclusion of c originating from ψ_2, then

$$q(\nu, \pi') = q(\nu, \pi)\{\alpha_1 \setminus \alpha_1', \ldots, \alpha_n \setminus \alpha_n', \beta_1 \setminus \beta_1', \ldots, \beta_m \setminus \beta_m'\} \vee$$
$$\vee\, q(\nu, \pi)\{\alpha_1 \setminus \alpha_1'', \ldots, \alpha_n \setminus \alpha_n'', \beta_1 \setminus \beta_1'', \ldots, \beta_m \setminus \beta_m''\}$$

because ν is contracted in π'.

If c is above the strong side of $\tilde{c}$, then eigenvariables of $\tilde{c}$ might be duplicated, as noted above. In that case, we obtain the new constraint formula of $\tilde{c}$ by replacing each $\beta_i \to t$ in $\mathcal{C}_{\tilde{c}}(\pi)$ with $\beta_i' \to t \vee \beta_i'' \to t$.

Now let $d = \varphi \to^* s$ be a valid derivation of $G(\pi)$. If no nonterminals belonging to c are used in d then all we have to do to obtain a valid derivation of $G(\pi')$ is replace each

β_i that occurs in d with β'_i. If, on the other hand, such nonterminals are used, then all of them must be produced from nonterminals of the end sequent due to the minimality of c. Let $\alpha_{i_1}, \ldots, \alpha_{i_m}$ be those nonterminals of c that occur in d and assume that each α_{i_j} is later replaced by a term t_j. Then either all of these terms are above μ' or all of them are above μ''. To see this, assume w.l.o.g. that α_{i_1} is later replaced by a term t_1 that introduces a quantifier in μ', but not in μ'' and α_{i_2} by a term t_2 for which the converse is true. Since d is valid, the atom $\alpha_{i_j} \to t_j$ in $\mathcal{C}(\pi)$ is assigned the value $\top$ by v_d and all other atoms beginning with α_{i_j} are assigned $\bot$, due to rigidity. $\mathcal{C}_c(\pi)$ is certainly of the form $\tilde{\mathcal{B}}' \vee \tilde{\mathcal{B}}''$. Since d is valid, either $v_d(\tilde{\mathcal{B}}') = \top$ or $v_d(\tilde{\mathcal{B}}'') = \top$; say the former w.l.o.g. But all α_{i_2}-atoms that occur in $\tilde{\mathcal{B}}'$ evaluate to $\bot$, which is a contradiction.

We now consider the case where all terms produced from the α_{i_j} introduce quantifiers in μ'. In this case, replacing all α_{i_j} in d with α'_{i_j} yields productions of $G(\pi')$. An analogous substitution applied to the c-nonterminals that are introduced by other end sequent nonterminals gives a new derivation d'. The derivation d might also contain some of the β_i. Since the β'_i and the β''_i have the same productions in P' as the β_i do in P, we can simply replace their productions as necessary.

Thus, we obtain a derivation d'' that consists of productions of $G(\pi')$; we now need to show that it is in fact valid. First of all, note that by construction, d'' obeys local rigidity. As for the constraint formula, it is clearly sufficient to show that $v_{d''}$ validates the various conjuncts of $\mathcal{C}(\pi')$.

- If $\tilde{c}$ is a cut with an eigenvariable among the β_i, say β_{i_0}, and β_{i_0} has an associated term t, then the atom $\beta_{i_0} \to t$ in $\mathcal{C}_{\tilde{c}}(\pi)$ is replaced with $\beta'_{i_0} \to t \vee \beta''_{i_0} \to t$ in $\mathcal{C}_{\tilde{c}}(\pi')$ and since $v_d(\mathcal{C}_{\tilde{c}}(\pi)) \leftrightarrow \top$, the same holds for $v_{d''}(\mathcal{C}_{\tilde{c}}(\pi'))$.
- Clearly, $v_{d''}(\mathcal{C}_{c''}(\pi')) = \top$ because none of the α''_i-productions are evaluated by $v_{d''}$.
- $v_{d''}(\mathcal{C}_{c'}(\pi')) = \top$ follows immediately from $v_d(\mathcal{C}_c(\pi)) = \top$.
- The constraint formulas of other cuts and the end sequent are easily seen to be valid under $v_{d''}$.

Conversely, suppose that we have a derivation d' of $G(\pi')$. The first thing we need to establish is that d' can only contain nonterminals of c' or c'', but not both. This is the case because there is no production that contains nonterminals of both and and $\mathcal{C}_{\mathrm{ES}}(\pi')$ forces us to choose either ψ'_2 or ψ''_2 in each derivation. We thus obtain a derivation d of $G(\pi)$ by replacing all $\alpha'_i, \beta'_i, \alpha''_i, \beta''_i$ with their original versions. This d does not violate rigidity due to the considerations above. As in the argument for the other direction, the satisfiability under d of the various parts of $\mathcal{C}$ follows readily from the satisfiability of the corresponding parts of $\mathcal{C}'$.

Now suppose that the contraction happens on the strong side of c. Reducing the contraction leaves us with two new cuts c', c'' whose cut formulas are both A_c. Let μ' and μ'' be the occurrences of A_c that serve as cut formulas for c' and c'' respectively. Each eigenvariable α of c introduces a quantifier in either μ' or μ'' and consequently belongs to either c' or c'' accordingly. Consequently, $EV(c) = EV(c') \dot{\cup} EV(c'')$, where either set on the right-hand side might be empty. Thus, let $EV(c) = \{\alpha_1, \ldots, \alpha_n\}$ and assume for the sake of simplicity that $EV(c') = \{\alpha_1, \ldots, \alpha_k\}$ and $EV(c'') = \{\alpha_{k+1}, \ldots, \alpha_n\}$.

The duplication of the left subproof ψ_1 has extensive effects on the grammar. We will discuss these effects separately for each $\tilde{c} \in \mathrm{QCuts}(\pi)$. First, if $\tilde{c}$ is below c, then c must be on the strong side of $\tilde{c}$ due to c's minimality. As a consequence, it is possible that there are eigenvariables of $\tilde{c}$ that are introduced within ψ_1. If γ is such an eigenvariable, then γ is duplicated, giving rise to eigenvariables γ' and γ''. Each such γ' and γ'' inherits the productions of γ in $G(\pi)$. The constraint formulas of $\tilde{c}$ changes in a straightforward manner,

by replacing $\gamma \to t$ with $\gamma' \to t \vee \gamma'' \to t$ for each γ that is duplicated. In the sequel, let $\{\gamma_1, \ldots, \gamma_l\}$ be all eigenvariables of the original proof duplicated in this manner.

Next, assume that $\tilde{c}$ is located in ψ_1. In this case, $\tilde{c}$ is replaced with two new cuts $\tilde{c}'$ and $\tilde{c}''$. If $\{\beta_1, \ldots, \beta_m\}$ are all eigenvariables that belong to such cuts, then clearly each of them is replaced by two new copies β_i' and β_i''. The productions of these duplicates work out to

$$P'_{\beta'} = P_{\beta_i}\{\bar{\beta} \setminus \bar{\beta}', \bar{\gamma} \setminus \bar{\gamma}'\},$$
$$P'_{\beta''} = P_{\beta_i}\{\bar{\beta} \setminus \bar{\beta}'', \bar{\gamma} \setminus \bar{\gamma}''\}$$

for each $i \in \{1, \ldots, m\}$. Similarly, $\tilde{c}'$ and $\tilde{c}''$ have the constraint formulas

$$\mathcal{C}_{\tilde{c}'} = \mathcal{C}_{\tilde{c}}\{\beta_1 \setminus \beta_1', \ldots, \beta_m \setminus \beta_m'\},$$
$$\mathcal{C}_{\tilde{c}''} = \mathcal{C}_{\tilde{c}}\{\beta_1 \setminus \beta_1'', \ldots, \beta_m \setminus \beta_m''\}$$

respectively. The final case to consider is that of c itself: The productions of the α_i in $G(\pi')$ work out to

$$P'_{\alpha_i} = P_{\alpha_i}[\beta_1 \setminus \beta_1', \ldots, \beta_m \setminus \beta_m', \gamma_1 \setminus \gamma_1', \ldots, \gamma_k \setminus \gamma_l'] \text{ for } i \leq k,$$
$$P'_{\alpha_i} = P_{\alpha_i}[\beta_1 \setminus \beta_1'', \ldots, \beta_m \setminus \beta_m'', \gamma_1 \setminus \gamma_1'', \ldots, \gamma_k \setminus \gamma_l''] \text{ for } i > k.$$

The constraint formula of c' can be obtained from $\mathcal{C}_c$ by replacing each literal $\alpha_i \to t$ that occurs in it with $\alpha_i \to t[\beta_1 \setminus \beta_1', \ldots, \beta_m \setminus \beta_m', \gamma_1 \setminus \gamma_1', \ldots, \gamma_k \setminus \gamma_k']$ (for $i \leq k$) or removing it (for $i > k$). An analogous transformation yields $\mathcal{C}_{c''}$. If $\tilde{c}$ is any other quantified cut, then $\tilde{c}$ is either within the strong side of c or on a different branch of the proof from c. The first case is impossible due to minimality of c and in the second case, $\tilde{c}$ is unaffected by the proof transformation.

The last thing that needs to be taken care of are the productions and constraint formula of the end sequent. Each production $z_i \to t$ is replaced by

$$z_i \to t[\beta_1 \setminus \beta_1', \ldots, \beta_m \setminus \beta_m', \gamma_1 \setminus \gamma_1', \ldots, \gamma_k \setminus \gamma_k'] \text{ and}$$
$$z_i \to t[\beta_1 \setminus \beta_1'', \ldots, \beta_m \setminus \beta_m'', \gamma_1 \setminus \gamma_1'', \ldots, \gamma_k \setminus \gamma_k''].$$

If t does not contain any β_i or γ_i, then both of these duplicates obviously coincide with the original production and it simply carries over to $G(\pi')$. As for $\mathcal{C}_{\mathrm{ES}}(\pi')$, there are formulas $\mathcal{B}_1, \ldots, \mathcal{B}_r$ such that

$$\begin{aligned}
\mathcal{C}_{\mathrm{ES}}(\pi) &= \mathcal{C}[\mathcal{B}_1, \ldots \mathcal{B}_r] \text{ and} \\
\mathcal{C}_{\mathrm{ES}}(\pi') &= \mathcal{C}[\mathcal{B}_1[\beta_1 \setminus \beta_1', \ldots, \beta_m \setminus \beta_m', \gamma_1 \setminus \gamma_1', \ldots, \gamma_k \setminus \gamma_k'] \vee \\
&\quad \vee \mathcal{B}_1[\beta_1 \setminus \beta_1'', \ldots, \beta_m \setminus \beta_m'', \gamma_1 \setminus \gamma_1'', \ldots, \gamma_k \setminus \gamma_k''], \\
&\quad \ldots \\
&\quad \mathcal{B}_r[\beta_1 \setminus \beta_1', \ldots, \beta_m \setminus \beta_m', \gamma_1 \setminus \gamma_1', \ldots, \gamma_k \setminus \gamma_k'] \vee \\
&\quad \vee \mathcal{B}_r[\beta_1 \setminus \beta_1, \ldots, \beta_m \setminus \beta_m'', \gamma_1 \setminus \gamma_1'', \ldots, \gamma_k \setminus \gamma_k'']].
\end{aligned}$$

Let d be a valid derivation of $G(\pi)$. If nonterminals of c occur in d, then due to the minimality of c they can only be introduced from nonterminals of the end sequent. Let $\alpha_{i_1}, \ldots, \alpha_{i_r}$ be those nonterminals of c that are used in d and are later replaced by terms $t_1, \ldots, t_r$. For each i_j, we replace the production $\alpha_{i_j} \to t_j$ with $\alpha_{i_j} \to t_j[\bar{\beta} \setminus \bar{\beta}', \bar{\gamma} \setminus \bar{\gamma}']$ if $i_j \leq k$ or $\alpha_{i_j} \to t_j[\bar{\beta} \setminus \bar{\beta}'', \bar{\gamma} \setminus \bar{\gamma}'']$ if $i_j > k$. Also, if $z_i \to t$ is a production of the end sequent in d, we replace it with $z_i \to t[\bar{\beta} \setminus \bar{\beta}', \bar{\gamma} \setminus \bar{\gamma}']$, obtaining a new derivation d'. This can lead to

d' containing both β_i' and β_i'' for some i, and similarly for the γ_i. Due to total rigidity, d uses at most one production for each β_i and γ_i and we can simply replace any such production by one or both of its two variants in the new grammar, according to whether one or both copies of the respective nonterminal occur in d'. We call the derivation obtained by this process d''.

As before, it is sufficient to show that d'' is totally rigid and validates the conjuncts of $\mathcal{C}(\pi')$. $v_{d''}(\mathcal{C}_{c'}) = \top$ because up to renaming, the literals of $\mathcal{C}_{c'}$ are a subset of those of $\mathcal{C}_c$ and $v_d(\mathcal{C}_c) = \top$. The satisfiability of $v_{d''}(\mathcal{C}_{c''})$ is shown in an analogous manner. The constraint formulas of all other cuts are similarly easy to deal with because they contain the same substitutions relative to their original counterparts as d'' does to d. $v_{d''}(\mathcal{C}_{\mathrm{ES}}(\pi')) = \top$ is immediately obvious.

Now suppose that we have a valid derivation d' of $G(\pi')$. First of all, there are some important conclusions to be drawn from the form of $\mathcal{C}_{\mathrm{ES}}(\pi')$: Let x, y be nonterminals of the end sequent such that x dominates y. If some production $x \to t(\bar{\alpha})$ is used in d', no production of y that is used in d' can contain any of the β_i' or γ_i' (or their $''$-versions), and vice versa. Moreover, if there is a production $x \to t_i(\bar{\beta}', \bar{\gamma}')$ in d', then productions $y \to t_j(\bar{\beta}'', \bar{\gamma}'')$ cannot occur in d', and analogously with the $'$- and $''$-nonterminals changed around. Since π is pruned, no term in ψ_2 contains two eigenvariables that introduce the same quantifier. These facts imply that we can simply replace all $'$- and $''$-nonterminals by their original versions without violating total rigidity. The argument that the resulting derivation d is valid then goes through just as in the previous cases. ◀

We can now finally prove the main result of this paper:

▶ **Theorem 16.** *Let π be a simple proof of $\Gamma \vdash \Delta$. Then $L(G(\pi))$ is an Herbrand set of $\Gamma \vdash \Delta$.*

Proof. By combining Lemmas 17, 19, 21, and 22. ◀

6 Conclusion

In this paper we have given a description of the Herbrand set induced by a proof with non-prenex Π_1 and Σ_1 cuts in terms of a tree grammar. This is a considerable extension of the previously existing work for prenex formulas [8] since the structure of sequent calculus proofs and the dynamics of cut-elimination changes significantly when non-prenex cuts are allowed. The central tool for this description are constraint grammars, which permit capturing the dependencies of the quantifier instantiations in the proof.

Applications of the connection between formal language theory and proof theory described in [8] for prenex Π_1 and Σ_1 cuts include results on Herbrand-confluence [12], cut-introduction [11, 10, 9], inductive theorem proving [4], and proof complexity [3]. In addition, this connection has recently been extended to prenex Π_2 and Σ_2 cuts [1]. In this line of research, this paper is the first to consider *non-prenex* formulas and thus opens the way for extending the above results and techniques to non-prenex cuts and induction formulas.

References

1 Bahareh Afshari, Stefan Hetzl, and Graham E. Leigh. Herbrand disjunctions, cut elimination and context-free tree grammars. to appear at Typed Lambda Calculi and Applications (TLCA) 2015.

2 Samuel R. Buss. On Herbrand's Theorem. In *Logic and Computational Complexity*, volume 960 of *Lecture Notes in Computer Science*, pages 195–209. Springer, 1995.

3 Sebastian Eberhard and Stefan Hetzl. Compressibility of finite languages by grammars. to appear at Descriptional Complexity of Formal Systems (DCFS) 2015, preprint available at `http://www.logic.at/people/hetzl/research/`.

4 Sebastian Eberhard and Stefan Hetzl. Inductive theorem proving based on tree grammars. *Annals of Pure and Applied Logic*, 166(6):665–700, 2015.

5 Philipp Gerhardy and Ulrich Kohlenbach. Extracting Herbrand Disjunctions by Functional Interpretation. *Archive for Mathematical Logic*, 44:633–644, 2005.

6 Willem Heijltjes. Classical proof forestry. *Annals of Pure and Applied Logic*, 161(11):1346–1366, 2010.

7 Jacques Herbrand. *Recherches sur la théorie de la démonstration*. PhD thesis, Université de Paris, 1930.

8 Stefan Hetzl. Applying Tree Languages in Proof Theory. In Adrian-Horia Dediu and Carlos Martín-Vide, editors, *Language and Automata Theory and Applications (LATA) 2012*, volume 7183 of *Lecture Notes in Computer Science*, pages 301–312. Springer, 2012.

9 Stefan Hetzl, Alexander Leitsch, Giselle Reis, Janos Tapolczai, and Daniel Weller. Introducing Quantified Cuts in Logic with Equality. In Stéphane Demri, Deepak Kapur, and Christoph Weidenbach, editors, *Automated Reasoning – 7th International Joint Conference, IJCAR*, volume 8562 of *Lecture Notes in Computer Science*, pages 240–254. Springer, 2014.

10 Stefan Hetzl, Alexander Leitsch, Giselle Reis, and Daniel Weller. Algorithmic introduction of quantified cuts. *Theoretical Computer Science*, 549:1–16, 2014.

11 Stefan Hetzl, Alexander Leitsch, and Daniel Weller. Towards Algorithmic Cut-Introduction. In *Logic for Programming, Artificial Intelligence and Reasoning (LPAR-18)*, volume 7180 of *Lecture Notes in Computer Science*, pages 228–242. Springer, 2012.

12 Stefan Hetzl and Lutz Straßburger. Herbrand-Confluence for Cut-Elimination in Classical First-Order Logic. In Patrick Cégielski and Arnaud Durand, editors, *Computer Science Logic (CSL) 2012*, volume 16 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 320–334. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2012.

13 Stefan Hetzl and Daniel Weller. Expansion trees with cut. preprint, available at `http://arxiv.org/abs/1308.0428`, 2013.

14 David Hilbert and Paul Bernays. *Grundlagen der Mathematik II*. Springer, 1939.

15 Markus Lohrey. Algorithmics on SLP-compressed strings: A survey. *Groups Complexity Cryptology*, 4(2):241–299, 2012.

16 Richard McKinley. Proof nets for Herbrand's Theorem. *ACM Transactions on Computational Logic*, 14(1):5:1–5:31, 2013.

17 Dale A. Miller. A compact representation of proofs. *Studia Logica*, 46(4):347–370, 1987.

18 V.P. Orevkov. Lower bounds for increasing complexity of derivations after cut elimination. *Zapiski Nauchnykh Seminarov Leningradskogo Otdeleniya Matematicheskogo Instituta*, 88:137–161, 1979.

19 Pavel Pudlák. The Lengths of Proofs. In Sam Buss, editor, *Handbook of Proof Theory*, pages 547–637. Elsevier, 1998.

20 Richard Statman. Lower bounds on Herbrand's theorem. *Proceedings of the American Mathematical Society*, 75:104–107, 1979.

21 Sebastian Zivota. Cuts Without Quantifier Alternations and Their Effect on Expansion Trees. Master's thesis, Vienna University of Technology, 2014.

Automata Theoretic Account of Proof Search

Aleksy Schubert*[1], Wil Dekkers[2], and Henk P. Barendregt[2]

1 **University of Warsaw, ul. Banacha 2, 02-097 Warsaw, Poland**
`alx@mimuw.edu.pl`
2 **Radboud University, Nijmegen, Faculty of Science, Postbus 9010, 6500 GL Nijmegen, The Netherlands**

Abstract

Techniques from automata theory are developed that handle search for inhabitants in the simply typed lambda calculus. The resulting method for inhabitant search, which can be viewed as proof search by the Curry-Howard isomorphism, is proven to be adequate by a reduction of the inhabitant existence problem to the emptiness problem for appropriately defined automata. To strengthen the claim, it is demonstrated that the latter has the same complexity as the former. We also discuss the basic closure properties of the automata.

1998 ACM Subject Classification E.1 Data structures, F.1.1 Models of computation, F.4.1 Mathematical logic, F.4.3 Formal languages

Keywords and phrases simple types, automata, trees, languages of proofs

Digital Object Identifier 10.4230/LIPIcs.CSL.2015.128

1 Introduction

A recent book on typed lambda calculi [1] contains a considerable number of graphical representations for constructing inhabitants of given types (or, by the Curry-Howard isomorphism, proofs for propositional intuitionistic logic). These follow the mechanics of the Ben-Yelles-Wajsberg algorithm [3, 9, 21], explain intuitively its operations, and materialise its mechanics for particular inputs. In this way, the runs of the algorithm become a particular compact data structure that can in itself, when defined formally, be subject to further computations, as finite automata are in automata theory.

However, it is not immediate how to turn the intuitive pictures into a formal notion. First of all, finite automata work on a finite alphabet, while λ-terms can contain bound variables from an infinite set. One approach available here is to restrict the language to a fixed set of first-order constants so that there is no need to introduce binders. This method was used in the work of Düdder et al. [6], where automata in this fashion were proposed for synthesis of programs in a simple functional language. Another approach would be to restrict the inhabitant search to a limited subset of the normal terms, that is sufficiently well distributed so that existence of a type inhabitant implies it is possible to find one of this form. Typically, *total discharge terms* are used for this purpose. In terms of this form, only one bound variable is needed for each subtype of the original type. Based upon this idea Takahashi et al [18] developed a context-free grammar approach to inhabitant search (which can be viewed as inhabitant search using tree automata due to known correspondence between grammars and tree automata).

* This work was partly supported by NCN grant DEC-2012/07/B/ST6/01532.

24th EACSL Annual Conference on Computer Science Logic (CSL 2015).
Editor: Stephan Kreutzer; pp. 128–143

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

These approaches have as natural limitation that they do not make it possible to recognise the collection of all inhabitants. For this a method is needed to deal with the infinite alphabet in the language. Automata that work with infinite alphabets were proposed by Kaminski and Francez [11] for strings and by Kaminski and Tan [12] for trees. These automata have, in addition to the standard control arranged through states, a fixed set of registers where data from an infinite set may be stored. Data elements stored in registers can be checked for equality with data elements from the input. This restricted check operation on an infinite domain makes it possible to work with such automata similarly to the usage of standard finite automata. Still, these automata do not fit well with the type inhabitation problem, as storing a fixed finite number of bound variable names is not sufficient to represent all potential normal inhabitants. To overcome this limitation we propose here a different notion of register, such that *a set* of data elements may be kept there and the check operation verifies whether a data element from the input *belongs* to the set. It turns out that this method recognises all trees that can be reasonably regarded as inhabitants of a particular type. Moreover, we show how it relates to earlier approaches, in which the *total discharge forms* are recognised.

Related work. Various kinds of finite automata have been proposed for dealing with semantics of the simply typed λ-calculus. The work of Salvati and Walukiewicz expresses the semantics through Krivine machines [15]. Another approach expresses semantics by description of β-reduction in the context of the higher-order matching problem (Ong and Tzevelekos [13], Stirling [17]). Another interesting related work goes in a different direction. Broda and Damas [4] proposed the formula-tree proof method, which partially realises the program of the current work, and concretisises the proof search procedure as a data structure. We believe that the automata theoretic view proposed here has the additional benefit of bridging proof theory with the rich theory of automata, enabling mutual influence.

Organisation of the paper. We fix the notation in Section 2. Next, we define our inhabitation machines in Section 3. This is continued by demonstration of the PSPACE-completeness of the emptiness problem for the machines in Section 4. We summarise the account in Section 5 by giving conclusions and showing the potential for further work.

2 Preliminaries

To make this paper self-contained we introduce some basic notation. In the automata theoretic setting it is convenient to use the notion of a *signature*, usually denoted by Σ, that is an indexed family of sets that contain elements called *symbols*. We sometimes abuse the notation by identifying Σ with $\bigcup \Sigma$ and write e.g. $a \in \Sigma$ for some symbol a in one of the members of Σ. The indices of the family are natural numbers and are called *arities*. The arity of a symbol f is written $\mathsf{arity}(f)$. For a natural number k we write $\overline{k}$ for the set $\{0, \ldots, k-1\}$. The set of all subsets of a given set A is written $P(A)$, and for the set of all finite subsets of A we write $P_{\mathsf{fin}}(A)$. Concatenation of two sequences π, π' of elements from some set A is written $\pi \cdot \pi'$. A special case here is when π' is a single symbol $i \in A$, then the concatenation is $\pi \cdot i$. The prefix order on sequences of natural numbers is written $\preceq$. A set C of finite sequences over $\mathbb{N}$ that is closed on the prefixes can be used as a domain of a tree. A *labelled tree* over L is a function $t : C \to L$ where L is called the *set of labels*. We write $\mathsf{dom}(t)$ for C. Elements of $\mathsf{dom}(t)$ are called *nodes in the tree* t. We write $t|_\pi$ for the subtree rooted at the node $\pi \in \mathsf{dom}(C)$, i.e. the tree with the domain $C' = \{\pi' \mid \pi \cdot \pi' \in \mathsf{dom}(t)\}$

and labelling t' defined as $t'(\pi') = t(\pi \cdot \pi')$ being t restricted to C'. Usually, the set of labels is a signature (flattened to $\bigcup \Sigma$) and then we assume that the tree respects the arity, i.e. if $\mathsf{arity}(t(\pi)) = n$ then $\pi \cdot i \in \mathsf{dom}(t)$ for $i \in \overline{n}$. For a function $f : A \to B$ we define its update $f[a \mapsto b]$ for $a \in A$ and $b \in B$ as $f[a \mapsto b](x) = f(x)$ for $x \neq a$ and $f[a \mapsto b](a) = b$.

3 Automata account of the inhabitation problem

In what follows we use a slightly modified exposition from [1]. The simply typed λ-calculus $\lambda_{\to}$ in the Church style is a language of expressions that have the following syntax expressed in *simplified syntax BNF*:

$$\begin{array}{lll} \mathbb{T} \ni & \tau & ::= \alpha \mid \tau_0 \to \tau_1 \\ \Lambda_{\to} \ni & M & ::= x^{\tau} \mid M_0 M_1 \mid \lambda x^{\tau}.M_0 \end{array}$$

This means that the parentheses are left implicit in the grammar above. The elements of $\mathbb{T}$ and $\Lambda_{\to}$ are called *types* and *terms*, respectively. We assume here that α are *type atoms* that are from an infinite, countable set A. We use metavariables σ, τ etc. for types. Term variables, noted x, y, F etc. are from an infinite countable set V. Compound expressions of the form x^{τ} are called *typed term variables* and the set that contains all of them is $\mathsf{V}^{\Lambda}_{\to}$. As usual we distinguish the set of free variables $\mathsf{FV}(M)$ and define it structurally over terms so that the binding operation is λ, and x^{τ} is bound in $\lambda x^{\tau}.M_0$. A term that has no occurrences of free variables is called *closed*. The λ-terms are identified up to α-conversion that makes it possible to rename bound variables. A *context*, usually written as Γ with possible ornaments, is a finite set of typed term variables.

We follow here a slightly non-standard take on contexts since we make it possible for a context to contain both x^{τ} and $x^{\tau'}$ for $\tau \neq \tau'$. Observe that this solution is not essential since the type makes the variables to be sufficiently distinct. One must only ensure that when a type erasure operation is performed, such two variables are made distinct, which can be done in different ways, e.g. by making the type a part of the variable name.

Terms of type τ in the context Γ, written $\Lambda^{\Gamma}_{\to}(\tau)$, are a family of sets defined as the smallest family that satisfies the conditions:

- $x^{\tau} \in \Lambda^{\Gamma}_{\to}(\tau)$ when $x^{\tau} \in \Gamma$,
- if $M_0 \in \Lambda^{\Gamma}_{\to}(\sigma \to \tau)$ and $M_1 \in \Lambda^{\Gamma}_{\to}(\sigma)$ then $M_0 M_1 \in \Lambda^{\Gamma}_{\to}(\tau)$,
- if $M_0 \in \Lambda^{\Gamma \cup \{x^{\sigma}\}}_{\to}(\sigma')$ then $\lambda x^{\sigma}.M_0 \in \Lambda^{\Gamma}_{\to}(\sigma \to \sigma')$ where $\sigma \to \sigma' = \tau$.

We often abbreviate $\Lambda^{\emptyset}_{\to}(\tau)$ as $\Lambda_{\to}(\tau)$.

Proof search procedures usually look for proof terms in normal form, i.e. ones that do not use a form of the *cut* rule. In the context of λ-calculi, the *cut* operation is represented as a beta redex. In case of $\lambda_{\to}$ in the Church style, this redex is

$$(\lambda x^{\tau}.M_0)M_1 \to_{\beta} M_0[x^{\tau} := M_1]$$

where $M_0[x^{\tau} := M_1]$ is understood as the term that results from M_0 by replacing all occurrences of the typed variable x^{τ} with M_1. This substitution, as usual, renames bound variables in M_0 so that no free variable in M_1 is captured by binding λ operators in M_0. The relation $\to_{\beta}$ is defined by syntax closure of the above mentioned redexes. The reflexive-transitive closure of $\to_{\beta}$ is written in $\to^*_{\beta}$. It is known that the relation $\to_{\beta}$ is strongly normalising, i.e. each sequence of terms $M_0, M_1, \ldots$, such that $M_i \to_{\beta} M_{i+1}$, has a finite number of elements ([1, Theorem 2.2.1]).

It is easy to see that *normal terms of type τ in the context* Γ, written $\Lambda^{\Gamma}_{n,\to}(\tau)$, are a family of sets defined as the smallest family that satisfies the conditions stated below. This definition uses a supplementary set $\Lambda^{\Gamma}_{s,\to}(\tau)$ (the letter 's' stands for 'spine' here).

- if $x^\tau \in \Gamma$ then $x^\tau \in \Lambda^\Gamma_{n,\to}(\tau)$ and $x^\tau \in \Lambda^\Gamma_{s,\to}(\tau)$,
- if $M_0 \in \Lambda^\Gamma_{s,\to}(\sigma \to \tau)$ and $M_1 \in \Lambda^\Gamma_{n,\to}(\sigma)$ then $M_0M_1 \in \Lambda^\Gamma_{s,\to}(\tau)$ and $M_0M_1 \in \Lambda^\Gamma_{n,\to}(\tau)$,
- if $M_0 \in \Lambda^{\Gamma\cup\{x^\sigma\}}_{n,\to}(\sigma')$ then $\lambda x^\sigma.M_0 \in \Lambda^\Gamma_{n,\to}(\sigma \to \sigma')$ where $\sigma \to \sigma' = \tau$.

A standard inductive argument shows the following proposition.

▶ **Proposition 1.** *If N is a subterm of $M \in \Lambda^\Gamma_{n,\to}(\tau)$ then $N \in \Lambda^{\Gamma'}_{n,\to}(\sigma)$ where $\Gamma \subseteq \Gamma'$ and all types in Γ' and σ are either subexpressions of τ or subexpressions of types in Γ.*

Note that the context Γ' may contain variables that do not occur in N.

The proof search when considered in the field of λ-calculi turns out to be, due to the Curry-Howard isomorphism, the search of inhabitants for types. Here is the precise formulation of the inhabitation problem.

▶ **Definition 2** (inhabitation problem)**.** The inhabitation problem for $\lambda_\to$ (or the decision problem for implicational fragment of propositional intuitionistic logic) is defined as follows
Input: A type τ.
Question: Is there a closed Church-style term M such that M has type τ?

▶ **Example 3.** Consider types $\mathbf{1} = 0 \to 0$, $\mathbf{2} = \mathbf{1} \to 0$ and $\mathbf{3} = \mathbf{2} \to 0$. A normal inhabitant of the type must have the form $\lambda F^{\mathbf{2}}.M_a$ where M_a is of type 0 (we use here the variable F instead of x to underline that it represents a functional). Next the only option we have is to use the typed variable $F^{\mathbf{2}}$, so $M_a = F^{\mathbf{2}}M_b$ where M_b is of type $\mathbf{1}$. Subsequently, we cannot use $F^{\mathbf{2}}$ so M_b must start with λ. Thus $M_b = \lambda y^0.M_c$ where M_c must be of type 0. We can now complete the process and let $M_c = y^0$, but we can continue the process by steps similar to the ones we used for M_a and obtain a sequence of terms

$$\lambda F^{\mathbf{2}}.F^{\mathbf{2}}(\lambda y^0.y^0), \quad \lambda F^{\mathbf{2}}.F^{\mathbf{2}}(\lambda y^0.F^{\mathbf{2}}(\lambda y^0_1.y^0_1)), \quad \lambda F^{\mathbf{2}}.F^{\mathbf{2}}(\lambda y^0.F^{\mathbf{2}}(\lambda y^0_1.F^{\mathbf{2}}(\lambda y^0_2.y^0_2))), \ldots \quad (1)$$

Note that this sequence does not exhaust the whole set of inhabitants of $\mathbf{3}$ since only the last variable of form y^0_i is used here while we have the liberty to use any of them.

In the following a special kind of terms called *total discharge terms* (or terms in Prawitz natural deduction style) [14, 19] is used as a technical device that helps in effective search for witnesses for non-emptiness. Suppose that we have an injection $\phi : \mathbb{T} \to \mathsf{V}$. Let us represent $\phi(\tau)$ as x_τ. We can now define a set of terms $\Lambda^{\mathsf{cnst}}_\to$ as the smallest subset of $\Lambda_\to$ such that

- all x^τ_τ belong to $\Lambda^{\mathsf{cnst}}_\to$,
- if M_0, M_1 belong to $\Lambda^{\mathsf{cnst}}_\to$ then M_0M_1 does,
- if M_0 belongs to $\Lambda^{\mathsf{cnst}}_\to$ and x^τ_τ is a typed variable then $\lambda x^\tau_\tau.M_0$ does.

The following proposition holds for $\lambda_\to$.

▶ **Proposition 4.** *For each context Γ if $\Lambda^\Gamma_\to(\tau) \neq \emptyset$ then $\Lambda^{\lfloor\Gamma\rfloor}_\to(\tau) \cap \Lambda^{\mathsf{cnst}}_\to \neq \emptyset$, where $\lfloor\Gamma\rfloor = \{x^\tau_\tau \mid y^\tau \in \Gamma\}$.*

Proof. Assume that some $M \in \Lambda^\Gamma_\to(\tau)$. We can now show by induction on the structure of M that there is a term $M' \in \Lambda^{\mathsf{cnst}}_\to$ that belongs to $\Lambda^{\lfloor\Gamma\rfloor}_\to(\tau)$. Details are left to the reader. ◀

▶ **Example 5.** The sequence of terms in (1) corresponds to the following sequence of terms in total discharge form.

$$\lambda F^{\mathbf{2}}.F^{\mathbf{2}}(\lambda y^0.y^0), \quad \lambda F^{\mathbf{2}}.F^{\mathbf{2}}(\lambda y^0.F^{\mathbf{2}}(\lambda y^0.y^0)), \quad \lambda F^{\mathbf{2}}.F^{\mathbf{2}}(\lambda y^0.F^{\mathbf{2}}(\lambda y^0.F^{\mathbf{2}}(\lambda y^0.y^0))), \ldots$$

Note that we use here y^0 only instead multiple $y^0_1, y^0_2 \ldots$ since in total discharge form only one variable for a type is allowed.

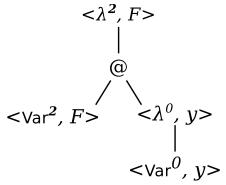


Figure 1 The tree representing the term $\lambda F^{\mathbf{2}}.F^{\mathbf{2}}(\lambda y^0.y^0)$.

3.1 Terms as trees

We identify terms in the Church style with trees in the following way. Let $\mathbb{T}^\sigma$ be the set of all subexpressions of σ. We define Σ^σ_T as the family with symbols $\{\mathsf{Var}^\tau \mid \tau \in \mathbb{T}^\sigma\}$ of arity 0, symbols $\{\lambda^\tau \mid \tau \in \mathbb{T}^\sigma\}$ of arity 1 and the symbol @ of arity 2. For a term $M \in \Lambda^\Gamma_{n,\to}(\tau)$ the tree t^M it corresponds to is defined inductively as follows:

- for $M = x^\tau$ it is a tree with a single node labelled with $\langle \mathsf{Var}^\tau, x\rangle$ and we write the tree as $\langle \mathsf{Var}^\tau, x\rangle$,
- for $M = M_0 M_1$ it is a tree t such that $t^M|_i = t^{M_i}$, for $i = 0, 1$ and $t(\varepsilon) = @$, we write the tree as $@(t^{M_0}, t^{M_1})$,
- for $M = \lambda x^\tau.M_0$ it is a tree t such that $t^M|_0 = t^{M_0}$ and $t(\varepsilon) = \langle \lambda^\tau, x\rangle$, we write the tree as $\langle \lambda^\tau, x\rangle(t^{M_0})$.

A tree that represents the first term in the sequence (1) in Example 3 is presented in Figure 1.

We can identify terms of λ-calculus with such trees since the sets are clearly in bijection one with the other. We introduce now the notion of α-conversion for trees and identify α-equivalent trees. First, let us define variable renaming.

▶ **Definition 6** (variable renaming)**.** We define inductively $t[y := x]^\tau$ in the following way

- $\langle \mathsf{Var}^{\tau'}, z\rangle[y := x]^\tau = \langle \mathsf{Var}^{\tau'}, z\rangle$ for $z \neq y$ or $\tau' \neq \tau$,
- $\langle \mathsf{Var}^\tau, y\rangle[y := x]^\tau = \langle \mathsf{Var}^\tau, x\rangle$,
- $@(t_0, t_1)[y := x]^\tau = @(t_0[y := x]^\tau, t_1[y := x]^\tau)$,
- $\langle \lambda^{\tau'}, z\rangle(t^{M_0})[y := x]^\tau = \langle \lambda^{\tau'}, z\rangle(t^{M_0}[y := x]^\tau)$ when $z \neq y, z \neq x$ or $\tau' \neq \tau$,
- $\langle \lambda^\tau, x\rangle(t^{M_0})[y := x]^\tau = \langle \lambda^\tau, z\rangle(t^{M_0}[x := z]^\tau[y := x]^\tau)$ where $z \neq y$ and $z \neq x$,
- $\langle \lambda^\tau, y\rangle(t^{M_0})[y := x]^\tau = \langle \lambda^\tau, y\rangle(t^{M_0})$.

The α-equivalence itself is defined as follows.

▶ **Definition 7** (α-equivalence)**.** For each variable x such that t does not have a free occurrence of the node $\langle \mathsf{Var}^\tau, x\rangle$ and each variable y we let $\langle \lambda^\tau, y\rangle(t) \equiv^0_\alpha \langle \lambda^\tau, x\rangle(t[y := x]^\tau)$. The closure of $\equiv^0_\alpha$ over the structure of trees is defined as $\equiv^s_\alpha$. The α conversion $\equiv_\alpha$ is defined as the reflexive-transitive closure of $\equiv^s_\alpha$.

As we can see, this definition is slightly non-standard since it makes it possible to use the same variable name in two different types as if they were two different variables. Indeed the variables are made different by their types. We admit that the term $\lambda x^{\alpha\to\alpha}.\lambda x^\alpha.x^{\alpha\to\alpha}x^\alpha$ is probably not legible for humans, but for machines it is as good as $\lambda x^{\alpha\to\alpha}.\lambda y^\alpha.x^{\alpha\to\alpha}y^\alpha$. Note that this does not work well for Curry-style terms where there is no way to distinguish different variables through their types. The advantage of this style is that it requires fewer variable names to represent λ-terms.

3.2 Inhabitation machines

The proof search associated with the inhabitation problem can be done in two fashions. In the generative fashion, we start with axioms and step-wise apply rules associated with

connectives bottom-up until the desired goal is reached. Another approach, called analytic fashion, consists in step-wise decomposition of the formula top-down until axioms are reached. Our definition of automaton follows the latter approach so it is a version of top-down tree automata.

▶ **Definition 8** (inhabitation machines)**.** A *(multiple assignment) inhabitation machine* (IM) $\mathcal{A}$ is a tuple $\langle \Sigma, N, Q, q_I, \mathcal{R}, \delta \rangle$ where Σ is a finite signature, N is an infinite set of data elements, Q is a finite set of states, $q_I \in Q$ is the initial state, $\mathcal{R}$ is a finite set (of register names), and $\delta \subseteq \Sigma \times Q \times P_{\mathsf{fin}}(\mathcal{R}) \times P_{\mathsf{fin}}(Q) \times P_{\mathsf{fin}}(\mathcal{R})$ is a set of rules written as $a, q, R \rightsquigarrow q_0, q_1, \ldots, q_{n-1}, W$ where $a \in \Sigma$, $q, q_0, \ldots, q_{n-1} \in Q$, and $R, W \in P_{\mathsf{fin}}(\mathcal{R})$.

The machine traverses labelled trees where the set of labels is $\Sigma \times N \cup \Sigma$. The arity of a pair $\langle a, x \rangle \in \Sigma \times N$ is the arity of a. We assume that all the rules respect the arity so that $\mathsf{arity}(a) = n$ in the rule above. In case all the rules are such that R, W are either empty or singleton sets, the machines are called *single assignment inhabitation machines.*

Observe that the transition rules of the machine do not include elements of the set N of data elements.

The operational semantics for such a machine is as follows. Configurations of $\mathcal{A}$ in a tree t are elements of $\mathsf{Config} = \mathsf{dom}(t) \times Q \times \mathsf{Reg}$ where $\mathsf{Reg} = \mathcal{R} \to P_{\mathsf{fin}}(N)$. Note that an element of Reg models a situation when a finite set of elements is held in a register of a given name from $\mathcal{R}$. Suppose we are in a configuration $\langle \pi, q, f \rangle$. Consider a rule

$$a, q, R \rightsquigarrow q_0, q_1, \ldots, q_{n-1}, W.$$

This rule is *applicable* in the configuration when

- $R = W = \emptyset$ and $t(\pi) = a$, or
- $t(\pi) = \langle a, x \rangle$ and $x \in f(r)$ for all $r \in R$.

As a result of such a rule the machine splits its control and moves to all n sons of the node π (recall that the arity must be respected both by the tree and by the rule) and for $i \in \overline{n}$ the *i-th resulting configuration* is $\langle \pi \cdot i, q_i, f^W_\downarrow \rangle$ where $f^W_\downarrow : \mathcal{R} \to P_{\mathsf{fin}}(N)$ is defined as

$$f^W_\downarrow(l) = \begin{cases} f(l) & \text{for } l \notin W, \\ f(l) \cup \{x\} & \text{for } l \in W. \end{cases} \tag{2}$$

Note that in case $W = \emptyset$ the condition in the second case of the definition is not possible so this pattern defines $f^W_\downarrow$ equal to f. We drop the superscript W whenever the set is clear from the context.

Whenever it does not lead to confusion we flatten the rules and instead of

$$a, q, \{i_0, \ldots, i_k\} \rightsquigarrow q_0, q_1, \ldots, q_{n-1}, \{j_0, \ldots, j_l\} \tag{3}$$

we write simply $a, q, i_0, \ldots, i_k \rightsquigarrow q_0, q_1, \ldots, q_{n-1}, j_0, \ldots, j_l$.

A *run* of a machine $\mathcal{A}$ on a tree t is a function $\mathfrak{r} : \mathsf{dom}(t) \to \mathsf{Config}$ that respects the rules of δ, i.e. for each node $\pi \in \mathsf{dom}(t)$ there is a rule $a, q, R \rightsquigarrow q_0, q_1, \ldots q_{n-1}, W \in \delta$ that is applicable in the configuration $\mathfrak{r}(\pi)$ and for each son i of the node π the configuration $\mathfrak{r}(\pi \cdot i)$ is the i-th resulting configuration of the rule.

We say that a machine $\mathcal{A}$ *accepts a tree t from a configuration* $\langle \pi, q, f \rangle$ when there is a correct run on $t|_\pi$ of $\mathcal{A}$ that starts with the configuration $\langle \varepsilon, q, f \rangle$. Let us define $f_I : \mathcal{R} \to P_{\mathsf{fin}}(N)$ so that $f_I(r) = \emptyset$ for all $r \in \mathcal{R}$. We say that the machine $\mathcal{A}$ *accepts a tree* t when there is a correct run of the machine that starts in $\langle \varepsilon, q_I, f_I \rangle$. The set of all trees t such that $\mathcal{A}$ accepts t is written $L(\mathcal{A})$.

▶ Remark (a version for the Curry style). A slightly different notion of machine is necessary to deal with terms in the Curry style. The definition of the resulting configuration must be modified. The state of the registers should change in a different way and the definition of the functions $f^W_\downarrow$ from (2) should be replaced with the following one

$$f^W_\downarrow(l) = \begin{cases} f(l)\backslash\{x\} & \text{for } l \notin W, \\ f(l) \cup \{x\} & \text{for } l \in W. \end{cases} \tag{4}$$

In this way, we maintain the interpretation that a particular variable name is active in a given scope for only one λ binder. The whole development of this paper could be redone for machines that use this version of register update. Its full examination is left for the full version of the paper.

Hereafter, a theorem is presented that relates inhabitation in $\lambda_\to$ and our machines. Before we formulate it, we define a crucial machine that is used there. The machine $\mathcal{A}_\tau$ is $\langle \Sigma^\tau_T, \mathsf{V}, Q, q_I, \mathcal{R}, \delta\rangle$ where

- $\Sigma^\tau_T, \mathsf{V}$ are defined as in Section 3.1,
- $Q = \{q_\sigma, q^s_\sigma \mid \sigma \text{ is a subexpression of } \tau\}$,
- $q_I = q_\tau$,
- $\mathcal{R}$ is the set of subexpressions of τ.

The rules of δ are as follows:

1. $\mathsf{Var}^\sigma, q_\sigma, \sigma \rightsquigarrow \emptyset$,
2. $\mathsf{Var}^\sigma, q^s_\sigma, \sigma \rightsquigarrow \emptyset$,
3. $@, q_\sigma, \emptyset \rightsquigarrow q^s_{\sigma'\to\sigma}, q_{\sigma'}, \emptyset$,
4. $@, q^s_\sigma, \emptyset \rightsquigarrow q^s_{\sigma'\to\sigma}, q_{\sigma'}, \emptyset$,
5. $\lambda^\sigma, q_{\sigma\to\sigma'}, \emptyset \rightsquigarrow q_{\sigma'}, \sigma$.

Note that these rules are such that the resulting machine is a single assignment IM.

▶ **Example 9.** Let us see how this construction works for the type $\mathbf{3}$. First note that subexpressions of $\mathbf{3}$ form the set $\mathcal{R}^{\mathbf{3}} = \{0, \mathbf{1}, \mathbf{2}, \mathbf{3}\}$. The automaton $\mathcal{A}_{\mathbf{3}} = \langle \Sigma^{\mathbf{3}}_T, \mathsf{V}, Q^{\mathbf{3}}, q_{\mathbf{3}}, \mathcal{R}^{\mathbf{3}}, \delta^{\mathbf{3}}\rangle$ where $\Sigma^{\mathbf{3}} = \{\mathsf{Var}^0, \mathsf{Var}^{\mathbf{1}}, \mathsf{Var}^{\mathbf{2}}, \mathsf{Var}^{\mathbf{3}}, \lambda^0, \lambda^{\mathbf{1}}, \lambda^{\mathbf{2}}, \lambda^{\mathbf{3}}, @\}$, $Q^{\mathbf{3}} = \{q_0, q_{\mathbf{1}}, q_{\mathbf{2}}, q_{\mathbf{3}}, q^s_0, q^s_{\mathbf{1}}, q^s_{\mathbf{2}}, q^s_{\mathbf{3}}\}$. The rules of $\delta^{\mathbf{3}}$ are

$\mathsf{Var}^0, q_0, 0 \rightsquigarrow \emptyset$,	$\mathsf{Var}^{\mathbf{1}}, q_{\mathbf{1}}, \mathbf{1} \rightsquigarrow \emptyset$,	$\mathsf{Var}^{\mathbf{2}}, q_{\mathbf{2}}, \mathbf{2} \rightsquigarrow \emptyset$,	$\mathsf{Var}^{\mathbf{3}}, q_{\mathbf{3}}, \mathbf{3} \rightsquigarrow \emptyset$,
$\mathsf{Var}^0, q^s_0, 0 \rightsquigarrow \emptyset$,	$\mathsf{Var}^{\mathbf{1}}, q^s_{\mathbf{1}}, \mathbf{1} \rightsquigarrow \emptyset$,	$\mathsf{Var}^{\mathbf{2}}, q^s_{\mathbf{2}}, \mathbf{2} \rightsquigarrow \emptyset$,	$\mathsf{Var}^{\mathbf{3}}, q^s_{\mathbf{3}}, \mathbf{3} \rightsquigarrow \emptyset$,
$@, q_0, \emptyset \rightsquigarrow q^s_{0\to 0}, q_0, \emptyset$,	$@, q_0, \emptyset \rightsquigarrow q^s_{\mathbf{1}\to 0}, q_{\mathbf{1}}, \emptyset$,	$@, q_0, \emptyset \rightsquigarrow q^s_{\mathbf{2}\to 0}, q_{\mathbf{2}}, \emptyset$,	
$@, q^s_0, \emptyset \rightsquigarrow q^s_{0\to 0}, q_0, \emptyset$,	$@, q^s_0, \emptyset \rightsquigarrow q^s_{\mathbf{1}\to 0}, q_{\mathbf{1}}, \emptyset$,	$@, q^s_0, \emptyset \rightsquigarrow q^s_{\mathbf{2}\to 0}, q_{\mathbf{2}}, \emptyset$,	
$\lambda^0, q_{0\to 0}, \emptyset \rightsquigarrow q_0, 0$,	$\lambda^{\mathbf{1}}, q_{\mathbf{1}\to 0}, \emptyset \rightsquigarrow q_0, \mathbf{1}$,	$\lambda^{\mathbf{2}}, q_{\mathbf{2}\to 0}, \emptyset \rightsquigarrow q_0, \mathbf{2}$.	

The construction presented here is a little bit not optimal as not all states and rules are reachable from the initial configuration. Yet, it is still simple in formulation and therefore convenient to handle in proofs.

Figure 2 presents the states of $\mathcal{A}_{\mathbf{3}}$ reachable from the initial configuration. An annotation next to an edge there indicates the alphabet symbol that is used to traverse it. For comparison with the machine in the book by Barendregt, Dekkers, and Statman [1, p. 36], the thick edges in the picture correspond directly to the edges there, while the thin edges should be collapsed to one edge labelled with F.

To demonstrate the operation of the automaton, we present here its run that witnesses that $\mathcal{A}_{\mathbf{3}}$ accepts the tree presented in Figure 1.

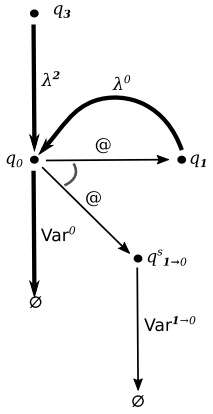


Figure 2 The automaton $\mathcal{A}_{\mathbf{3}}$ after removing non-reachable states.

1. $\langle \varepsilon, q_{\mathbf{3}}$, | 0 | **1** | **2** | **3** | $\rangle$ — We start at the root position in the initial state and with empty registers. The only possible rule to use is $\lambda^{\mathbf{2}}, q_{\mathbf{2}\to 0}, \emptyset \rightsquigarrow q_0, \mathbf{2}$.
2. $\langle 0, q_0$, | 0 | **1** | **2** | **3** | $\rangle$ | | | F | | — The register **2** was filled with a variable (F). We cannot apply the rule $\mathsf{Var}^0, q_0, 0 \rightsquigarrow \emptyset$ since the register 0 is empty. The only rules that remain are $@, q_0, \emptyset \rightsquigarrow q^s_\sigma, q_0, \emptyset$ where $\sigma \in \{0 \to 0, \mathbf{1} \to 0, \mathbf{2} \to 0\}$. After a while of analysis we can see that options where $\sigma \neq \mathbf{1} \to 0$ cannot lead to a successful computation. Thus, we follow the rule with $\sigma = \mathbf{1} \to 0$ and the computation forks to points 3. and 4. below.
3. $\langle 0, q^s_{\mathbf{1}\to 0},$ | 0 | **1** | **2** | **3** | $\rangle$ | | | F | | — Since $\mathbf{1} \to 0 = \mathbf{2}$ and the register **2** is not empty, we can apply the rule $\mathsf{Var}^{\mathbf{2}}, q^s_{\mathbf{2}}, \mathbf{2} \rightsquigarrow \emptyset$ and successfully terminate this branch of computation.
4. $\langle 0, q_{\mathbf{1}}$, | 0 | **1** | **2** | **3** | $\rangle$ | | | F | | — The register **1** is empty so we cannot apply the rule $\mathsf{Var}^{\mathbf{1}}, q_{\mathbf{1}}, \mathbf{1} \rightsquigarrow \emptyset$. Therefore, the only option is to use $\lambda^0, q_{0\to 0}, \emptyset \rightsquigarrow q_0, 0$ here and come back to q_0.
5. $\langle 0, q_0$, | 0 | **1** | **2** | **3** | $\rangle$ | y | | F | | — The register 0 is no longer empty so we can apply this time the rule $\mathsf{Var}^0, q_0, 0 \rightsquigarrow \emptyset$, which concludes the run of the automaton.

In step 5. we could use the rule with @ as in the step 2. and get into another turn of the loop visible in Figure 2. Looping there makes it possible to obtain trees representing other terms from (1) on page 131.

▶ **Theorem 10.** *For each type τ the language $L(\mathcal{A}_\tau)$ is the set of normal forms that are closed inhabitants of τ.*

Proof. Given a state of registers $f \in \mathsf{Reg}$ we can define a context Γ^f as $\Gamma^f = \{x^\sigma \mid x \in f(\sigma)\}$. Similarly, given a context Γ we can define a state of registers $f^\Gamma : \mathcal{R} \to P_{\mathsf{fin}}(\mathsf{V})$ determined as $f^\Gamma(\sigma) = \{x^\sigma \mid x^\sigma \in \Gamma\}$ where σ is a subexpression of τ. We have now the following fact:

1. Let σ be a subexpression of τ. If Γ is a context that contains only elements of the form $x^{\sigma'}$ where σ' is a subexpression of τ and $M \in \Lambda^\Gamma_{n,\to}(\sigma)$ ($M \in \Lambda^\Gamma_{s,\to}(\sigma)$) then there is a tree t^M such that $\mathcal{A}_\tau$ accepts t^M from a configuration $\langle \varepsilon, q_\sigma, f^\Gamma \rangle$ ($\langle \varepsilon, q^s_\sigma, f^\Gamma \rangle$, respectively).
2. If $\mathcal{A}_\tau$ accepts a tree t from a configuration $\langle \varepsilon, q_\sigma, f \rangle$ ($\langle \varepsilon, q^s_\sigma, f \rangle$) then there is a term $M \in \Lambda^{\Gamma^f}_{n,\to}(\sigma)$ ($M \in \Lambda^{\Gamma^f}_{s,\to}(\sigma)$, respectively), where M is such that $t = t^M$.

The proof of (1) is by induction over the structure of M

In case $M = x^\sigma$, we observe that $x^\sigma \in \Lambda^\Gamma_{n,\to}(\sigma)$ is possible only when $x^\sigma \in \Gamma$. This implies, as σ is a subexpression of τ, that $x^\sigma \in f^\Gamma(\sigma)$. As a result $\mathcal{A}_\tau$ accepts the tree $\langle \mathsf{Var}^\sigma, x\rangle$ from the configuration $\langle \varepsilon, q_\sigma, f^\Gamma\rangle$ through the rule $\mathsf{Var}^\sigma, q_\sigma, \sigma \rightsquigarrow \emptyset$. Similar argument applies for $x^\sigma \in \Lambda^\Gamma_{s,\to}(\sigma)$, but we have to use the rule $\mathsf{Var}^\sigma, q^s_\sigma, \sigma \rightsquigarrow \emptyset$.

In case $M = M_0M_1$, we observe that $M_0M_1 \in \Lambda^\Gamma_{n,\to}(\sigma)$ is possible only when $M_0 \in \Lambda^\Gamma_{s,\to}(\sigma' \to \sigma)$ and $M_1 \in \Lambda^\Gamma_{n,\to}(\sigma')$. By Proposition 1, the types $\sigma' \to \sigma$ and σ' are subexpressions of τ. Note that by the induction hypothesis we obtain a tree t^{M_0} such that $\mathcal{A}_\tau$ accepts it from the configuration $\langle \varepsilon, q^s_{\sigma'\to\sigma}, f^\Gamma\rangle$ through a run $\mathfrak{r}_0$ and a tree t^{M_1} such that $\mathcal{A}_\tau$ accepts it from the configuration $\langle \varepsilon, q_{\sigma'}, f^\Gamma\rangle$ through a run $\mathfrak{r}_1$. Let us construct a tree $t^M = @(t^{M_0}, t^{M_1})$ and a run $\mathfrak{r}$ over t^M such that $\mathfrak{r}(\varepsilon) = \langle \varepsilon, q_\sigma, f^\Gamma\rangle$, $\mathfrak{r}(0\cdot\pi) = \langle 0\cdot\pi, q', f'\rangle$ where $\mathfrak{r}_0(\pi) = \langle \pi, q', f'\rangle$, and $\mathfrak{r}(1\cdot\pi) = \langle 1\cdot\pi, q', f'\rangle$ where $\mathfrak{r}_1(\pi) = \langle \pi, q', f'\rangle$. It is easy to see that the rule $@, q_\sigma, \emptyset \rightsquigarrow q^s_{\sigma'\to\sigma}, q_{\sigma'}, \emptyset$ is applicable in the root node of t and for other nodes the function $\mathfrak{r}$ respects δ as $\mathfrak{r}_0$ and $\mathfrak{r}_1$ did.

Similar argument applies for $M_0M_1 \in \Lambda^\Gamma_{s,\to}(\sigma)$, but we have to use the rule $@, q^s_\sigma, \emptyset \rightsquigarrow q^s_{\sigma'\to\sigma}, q_{\sigma'}, \emptyset$.

In case $M = \lambda x^{\sigma'}.M_0$ we observe that $\lambda x^{\sigma'}.M_0 \in \Lambda^\Gamma_{n,\to}(\sigma' \to \tau')$ is possible only when $M_0 \in \Lambda^\Gamma_{n,\to}(\tau')$. By the induction hypothesis we obtain a tree t^{M_0} such that $\mathcal{A}_\tau$ accepts it from the configuration $\langle \varepsilon, q_{\tau'}, f^{\Gamma, x^{\sigma'}}\rangle$ through a run $\mathfrak{r}_0$. Let us consider the tree $t^M = \langle \lambda^{\sigma'}, x\rangle(t^{M_0})$ and a run $\mathfrak{r}$ over t such that $\mathfrak{r}(\varepsilon) = \langle \varepsilon, q_{\sigma'\to\tau'}, f^\Gamma\rangle$, and $\mathfrak{r}(0\cdot\pi) = \langle 0\cdot\pi, q', f'\rangle$ where $\mathfrak{r}_0(\pi) = \langle \pi, q', f'\rangle$. It is easy to see that the rule $\lambda^{\sigma'}, q_{\sigma'\to\tau'}, \emptyset \rightsquigarrow q_{\tau'}, \sigma'$ is applicable in the root node of t^M, and for other nodes the function $\mathfrak{r}$ respects δ as $\mathfrak{r}_0$ did.

The proof of (2) is by induction over the structure of t

In case $t = \langle \mathsf{Var}^{\tau'}, x\rangle$ and $\mathcal{A}_\tau$ accepts it from a configuration $\langle \varepsilon, q_\sigma, f\rangle$ ($\langle \varepsilon, q^s_\sigma, f\rangle$) then it can happen only because a rule of the form

$$\mathsf{Var}^\sigma, q_\sigma, \sigma \rightsquigarrow \emptyset \qquad (\text{or } \mathsf{Var}^\sigma, q^s_\sigma, \sigma \rightsquigarrow \emptyset)$$

was used. Such a rule is applicable only when $\tau' = \sigma$ and the register σ contains x. As a result x^σ is in Γ^f. This means $x^\sigma \in \Lambda^{\Gamma^f}_{n,\to}(\sigma)$ ($x^\sigma \in \Lambda^{\Gamma^f}_{s,\to}(\sigma)$, respectively) and thus the set is not empty.

In case $t = @(t^0, t^1)$ and $\mathcal{A}_\tau$ accepts t from a configuration $\langle \varepsilon, q_\sigma, f\rangle$ ($\langle \varepsilon, q^s_\sigma, f\rangle$) then it can happen only because a rule of the form

$$@, q_\sigma, \emptyset \rightsquigarrow q^s_{\sigma'\to\sigma}, q_{\sigma'}, \emptyset \qquad (\text{or } @, q^s_\sigma, \emptyset \rightsquigarrow q^s_{\sigma'\to\sigma}, q_{\sigma'}, \emptyset)$$

was used. The machine $\mathcal{A}_\tau$ accepts the tree t^0 from the configuration $\langle \varepsilon, q^s_{\sigma'\to\sigma}, f\rangle$ and $\mathcal{A}_\tau$ accepts t^1 from the configuration $\langle \varepsilon, q_{\sigma'}, f\rangle$. By the induction hypothesis we obtain that some $M_0 \in \Lambda^{\Gamma^f}_{s,\to}(\sigma' \to \sigma)$ and $M_1 \in \Lambda^{\Gamma^f}_{n,\to}(\sigma')$. As a result $M_0M_1 \in \Lambda^{\Gamma^f}_{n,\to}(\sigma)$ (or $M_0M_1 \in \Lambda^{\Gamma^f}_{n,\to}(\sigma)$, respectively).

In case $t = \langle \lambda^{\sigma'}, x\rangle(t^0)$ and $\mathcal{A}_\tau$ accepts t from a configuration $\langle \varepsilon, q_\sigma, f\rangle$ ($\langle \varepsilon, q^s_\sigma, f\rangle$) then it can happen only because a rule of the form

$$\lambda^{\sigma'}, q_{\sigma'\to\tau'}, \emptyset \rightsquigarrow q_{\tau'}, \sigma'$$

was used, where $\sigma' \to \tau' = \sigma$. The machine $\mathcal{A}_\tau$ accepts t^0 from the configuration $\langle \varepsilon, f', q_{\tau'}\rangle$ where $f' = f[\sigma' \mapsto f(\sigma') \cup \{x\}]$. By the induction hypothesis we obtain that some $M_0 \in \Lambda^{\Gamma^{f'}}_{n,\to}(\tau')$, which gives us that $\lambda x^{\sigma'}.M_0 \in \Lambda^{\Gamma^f}_{n,\to}(\sigma' \to \tau') = \Lambda^{\Gamma^f}_{n,\to}(\sigma)$.

A direct check verifies that the terms M constructed above have the property that $t = t^M$.

We can apply the proven above fact to the subexpression $\tau' = \tau$ and obtain the desired conclusion of Theorem 10. ◀

The proof of the theorem above easily generalises to the formulation that involves open terms as follows – given a fixed set Γ of free variables, type τ the language $L(A^{\Gamma}_{\tau})$ is the set of normal forms that are closed inhabitants of τ with free variables in Γ. It is simply enough to start the automaton with registers appropriately filled with variables from Γ.

Although a precise account of the remark below goes beyond the scope of this paper, it is worth observing that we could omit from the construction the (spine) states of the form q^s_σ and we would still obtain representations of typable terms. These terms would not need to be in normal form, though. Still, we would not be able to obtain all typable terms as we are limited by the finite number of registers that hold variables of types being subexpressions of the original type. Notably, this kind of restriction is natural in certain scenarios, in particular non-normal accessible terms considered in the decidability proofs for various versions of the higher-order matching problem could be accepted by our machines.

3.3 Invariance of α-conversion

The machines accept trees that are constructed from a particular set of variables. Still, λ-terms are understood up to renaming of bound variables, i.e. α-conversion. To establish the connection with terms rather than their α-representants we need to establish that the languages of trees accepted by the IM's defined before the proof of Theorem 10 cannot separate two different α-equivalent trees. Let us start with a definition which are the machines of interest here.

▶ **Definition 11** (variable consistent IM's)**.** Let $\mathcal{A} = \langle \Sigma^{\tau}_{T}, \mathsf{V}, Q, q_I, \mathcal{R}, \delta\rangle$ where $\mathcal{R}$ is the set of subexpressions of τ. We say that $\mathcal{A}$ is *a variable consistent IM* when all its rules with symbols Var^{σ} have the form $\mathsf{Var}^{\sigma}, q, \sigma \rightsquigarrow W$ for some $\sigma \in \mathcal{R}$.

▶ **Proposition 12.** *If t_1, t_2 are trees representing λ-terms such that $t_1 \equiv^0_\alpha t_2$ and $\mathcal{A}$ is a variable consistent IM that accepts t_1 from a configuration $\langle \varepsilon, q, f\rangle$ then it accepts t_2 from the same configuration.*

Proof. Observe that $t_1 \equiv^0_\alpha t_2$ means that $t_1 = \langle \lambda^{\tau}, y\rangle(t)$ and $t_2 = \langle \lambda^{\tau}, x\rangle(t[y := x]^{\tau})$. Let $\mathcal{A} = \langle \Sigma^{\sigma}_{T}, \mathsf{V}, Q, q_I, \mathcal{R}, \delta\rangle$. The proof is by induction over the size of the tree t.

In case $t = \langle \mathsf{Var}^{\tau'}, z\rangle$ we observe that $\mathcal{A}$ accepts t_1 from the configuration $\langle \varepsilon, q, f\rangle$, it must be the case that two rules are used to accomplish this

$$\lambda^{\tau}, q_1, R \rightsquigarrow q_2, W, \qquad \mathsf{Var}^{\tau'}, q_2, R' \rightsquigarrow W' \tag{5}$$

where $R, W, R, W' \in P_{\mathsf{fin}}(\mathcal{R})$.

We have now two subcases, (a) $\tau = \tau'$ with $z = y$, (b) $\tau = \tau'$ with $z \neq y$ or $\tau \neq \tau'$. In case (a), we have $t_1 = \langle \lambda^{\tau}, y\rangle(\langle \mathsf{Var}^{\tau}, y\rangle)$, $t_2 = \langle \lambda^{\tau}, x\rangle(\langle \mathsf{Var}^{\tau}, x\rangle)$. By a direct check we can verify that the same two rules can be used to accept t_2. The only non-trivial case is when $R' \neq \emptyset$, and then the check for presence of element in a register $i \in R'$ in both cases is positive since either $i \notin W$ and then in both cases $f(i)$ is the same or $i \in W$ and then in both cases the variable being checked is exactly the one that was added.

In case (b), we observe first that the case $\tau \neq \tau'$ is impossible in variable consistent IM's. Thus, we obtain that $t_1 = \langle \lambda^{\tau}, y\rangle(\langle \mathsf{Var}^{\tau}, z\rangle)$, $t_2 = \langle \lambda^{\tau}, x\rangle(\langle \mathsf{Var}^{\tau}, z\rangle)$. In this case, $W \cap R = \emptyset$ as otherwise the presence of z in any register of the intersection would mean $z = y$. This implies that only registers that were not modified by the rule with λ^{τ} can be checked and this makes the rules in (5) trivially accept t_2.

In case $t = @(t^0, t^1)$, we observe that $\mathcal{A}$ accepts t_1 with a run that starts with the rules

$$\lambda^{\tau}, q_1, R \rightsquigarrow q_2, W, \qquad @, q_2, \emptyset \rightsquigarrow q'_0, q'_1, \emptyset.$$

Note that once the machine is started in the initial configuration $\langle \varepsilon, q, f\rangle$ where $q = q_1$, it moves to two configurations $\langle i, q_i, f_\downarrow\rangle$ for $i = 0, 1$ where $f_\downarrow = f$ in case $W = \emptyset$ or $f_\downarrow = f[r_1 \mapsto f(r_1) \cup \{y\}] \cdots [r_l \mapsto f(r_l) \cup \{y\}]$ when $W = \{r_1, \ldots, r_l\}$. We accept t^i from configurations $\langle \varepsilon, q'_i, f_\downarrow\rangle$ for $i = 0, 1$. We take now two mild modifications $\mathcal{A}_i$ for $i = 0, 1$ of $\mathcal{A}$ obtained by adding the transition

$$\lambda^\tau, q_\bullet, \emptyset \rightsquigarrow q'_i, W$$

respectively, where $q_\bullet$ is a fresh state. We can directly verify that for $\mathcal{A}_i$ with the configuration $\langle \varepsilon, q_\bullet, f\rangle$ at the root of $t^i_1 = \langle \lambda^\tau, y\rangle(t^i)$, the resulting configuration is $\langle 0, q'_i, f_\downarrow\rangle$, and this configuration accepts t^i_1 provided that the machine accepts t^i from $\langle \varepsilon, q'_i, f\rangle$ for $i = 0, 1$. This is guaranteed by the fact that this holds for $\mathcal{A}$ and each its run is also a run of $\mathcal{A}_i$. We can now use the induction hypothesis to verify that $\mathcal{A}_i$ accept $t^i_2 = \langle \lambda^\tau, x\rangle(t^i[y := x]^\tau)$ from respective configurations for $i = 0, 1$. Note that the runs arrive at the configurations $\langle 0, q'_i, f_\downarrow\rangle$ respectively. They can be then turned to runs that accept $t^i[y := x]^\tau$ from $\langle \varepsilon, q'_i, f\rangle$. As the initial state does not occur on the right-hand sides of the rules they are actually runs of $\mathcal{A}$ too. Since the rule $\lambda^\tau, q_1, R \rightsquigarrow q_2, W$ transforms the initial configuration to a tuple $\langle i, q'_i, f_\downarrow\rangle$ and these are accepting when $\langle \varepsilon, q'_i, f_\downarrow\rangle$ are accepting for t_i, we obtain our conclusion that $\mathcal{A}$ accepts $\langle \lambda^\tau, x\rangle(t[y := x]^\tau)$.

In case $t = \langle \lambda^\tau, x\rangle(t)$ the proof is similar as in the previous case. The details are left to the reader. ◀

In case the machines are not variable consistent, we can find two trees, namely $t_1 = \langle \lambda^\tau, y\rangle(\langle \mathsf{Var}^\sigma, z\rangle)$ and $t_2 = \langle \lambda^\tau, x\rangle(\langle \mathsf{Var}^\sigma, z\rangle)$, that are in the relation $\equiv^0_\alpha$ but t_1 is accepted by a machine that rejects t_2.

▶ **Proposition 13.** *For any variable consistent IM $\mathcal{A}$, if $t_1 \equiv^s_\alpha t_2$ and $t_1 \in L(\mathcal{A})$ then $t_2 \in L(\mathcal{A})$.*

Proof. The proof is by a straightforward induction over the structure of t_1. Details are left to the reader. ◀

▶ **Theorem 14** (invariance of α-conversion)**.** *For any variable consistent IM $\mathcal{A}$, if $t_1 \equiv_\alpha t_2$ and $t_1 \in L(\mathcal{A})$ then $t_2 \in L(\mathcal{A})$.*

Proof. The proof is by induction on the number n of $\equiv^s_\alpha$ steps to obtain $t_1 \equiv_\alpha t_2$. The case of 0 steps is trivial since then $t_1 = t_2$. In case $n > 0$ we have t'_2 such that $t_1 \equiv_\alpha t'_2 \equiv^s_\alpha t_2$ and $t_1 \equiv_\alpha t'_2$ requires less than n steps of $\equiv^s_\alpha$. We obtain that $t'_2 \in L(\mathcal{A})$ by Proposition 13 and then $t_1 \in L(\mathcal{A})$ by the induction hypothesis. ◀

3.4 Closure properties

The advantage of automata is that they make it possible to easily give constructs for the sum or intersection of languages. This is done through closure constructions. We present these for the (multiple assignment) IM's proposed here.

▶ **Theorem 15.** *For all tree languages L_1, L_2 over a signature Σ if there are IM's $\mathcal{A}_i$ such that $L_i = L(\mathcal{A}_i)$ for $i = 1, 2$ then*

1. *there is a machine $\mathcal{A}$ such that $L(\mathcal{A}) = L_1 \cup L_2$,*
2. *there is a machine $\mathcal{A}$ such that $L(\mathcal{A}) = L_1 \cap L_2$.*

Proof. Let us first assume that $\mathcal{A}_i = \langle \Sigma, N, Q_i, q_{i,I}, \mathcal{R}_i, \delta_i \rangle$ for $i = 1, 2$ where $Q_1 \cap Q_2 = \emptyset$ and $\mathcal{R}_1 \cap \mathcal{R}_2 = \emptyset$. This assumption does not weaken our proof.

For the proof of (1) we define the machine whose states are the sum of states from $\mathcal{A}_1$ and $\mathcal{A}_2$ with registers $\mathcal{R}_1 \cup \mathcal{R}_2$. In addition the machine has a fresh initial state from which one can move non-deterministically either to states of $\mathcal{A}_1$ or to states of $\mathcal{A}_2$ and then continue the run according to the set of rules from the chosen this way machine. More precisely, $\mathcal{A} = \langle \Sigma, N, Q', q'_I, \mathcal{R}', \delta' \rangle$ where $Q' = Q_1 \cup Q_2 \cup \{q'_I\}$, q'_I is a fresh state, $\mathcal{R}' = \mathcal{R}_1 \cup \mathcal{R}_2$, and at last

$$\begin{aligned}\delta' = \delta_1 \cup \delta_2 \cup\ & \{a, q'_I, R \rightsquigarrow q_0, q_1, \ldots, q_{n-1}, W \mid a, q_{1,I}, R \rightsquigarrow q_0, q_1, \ldots, q_{n-1}, W \in \delta_1\} \cup \\ & \{a, q'_I, R \rightsquigarrow q_0, q_1, \ldots, q_{n-1}, W \mid a, q_{2,I}, R \rightsquigarrow q_0, q_1, \ldots, q_{n-1}, W \in \delta_2\}.\end{aligned}$$

The details of demonstration that this machine indeed recognises $L_1 \cup L_2$ are left to the reader.

For the proof of (2) we define the machine whose states are the product of the states from $\mathcal{A}_1$ and $\mathcal{A}_2$ with registers $\mathcal{R}_1 \cup \mathcal{R}_2$. The resulting machine simulates a run of $\mathcal{A}_1$ on the first coordinate and a run of $\mathcal{A}_2$ on the second one. When a set of registers R_1 is read in a rule of $\mathcal{A}_1$ and R_2 is read in a rule of $\mathcal{A}_2$ we combine them in a rule of the resulting machine by taking $R_1 \cup R_2$. Similarly for registers to write to. In this way, each time a set of registers is checked for presence of the current element of data all the registers in the rule from $\mathcal{A}_1$ are checked as well as ones for the rule from $\mathcal{A}_2$. Similarly for writes. In more detail the resulting machine is $\mathcal{A} = \langle \Sigma, N, Q', q'_I, \mathcal{R}', \delta' \rangle$ where $Q' = Q_1 \times Q_2$, $q_I = \langle q_{1,I}, q_{2,I} \rangle$, $\mathcal{R}' = \mathcal{R}_1 \cup \mathcal{R}_2$, and at last

$$\begin{aligned}\delta' = \{a, \langle q_{1,0}, q_{2,0} \rangle, R_1 \cup R_2 \rightsquigarrow \langle q_{1,1}, q_{2,1} \rangle, \langle q_{1,2}, q_{2,2} \rangle, \ldots \langle q_{1,n}, q_{2,n} \rangle, W_1 \cup W_2 \mid \\ a, q_{1,0}, R_1 \rightsquigarrow q_{1,1}, q_{1,2}, \ldots q_{1,n}, W_1 \in \delta_1, \text{ and} \\ a, q_{2,0}, R_2 \rightsquigarrow q_{2,1}, q_{2,2}, \ldots q_{2,n}, W_2 \in \delta_2\}.\end{aligned}$$

The details of demonstration that this machine recognises $L_1 \cap L_2$ are left to the reader. ◀

The constructions above use multiple assignments. An observant reader may have spotted that the proof of Theorem 10 requires only singleton sets in rules of machines. As a result, the machines there are single assignment IM's. It is an open question if the automata are closed on intersection. Therefore, we decided to introduce to the general model multiple register manipulations in the fashion of *multiple assignment* automata considered by Kaminski and Francez [11] where the closure can be obtained as above.

An immediate application of the above closure properties is the extension of the language of closed terms typed in the simply typed λ-calculus to the calculus of intersection types of rank 1 [20] or sum types of rank 1 [8].

4 The emptiness problem

To see how the design of the machines fits the inhabitation problem we show that the emptiness problem for the machines has the same complexity as the one for $\lambda_{\rightarrow}$. For this, we need to introduce another kind of automata for which the decidability of the emptiness problem can be dealt with more straightforwardly. Although we do not explore this connection in detail, the automata can be viewed as a reformulation of the term recognition by grammars proposed by Takahashi et al. [18].

We give here a construction that works for single assignment IM's with yet another restriction. This restriction covers the machines defined for the proof of Theorem 10. The general case requires more work and its demonstration would depart too much from the topic of inhabitation for the simply typed λ-calculus.

▶ **Definition 16** (one operation machines). A single assignment IM $\mathcal{A}$ is a *one operation machine* when its rules have form $a, q, R \rightsquigarrow q_0, \ldots, q_{n-1}, W$ where at most one of the sets R, W is non-empty.

▶ **Definition 17** (binary automata). A *binary automaton* (BA) $\mathcal{A}$ is a tuple $\langle \Sigma, Q, q_I, \mathcal{R}, \delta \rangle$ where Σ is a finite signature, Q is a finite set of states, $q_I \in Q$ is the initial state, $\mathcal{R}$ is a finite set (of available register names), and δ is a set of rules of the form $a, q, R \rightsquigarrow q_0, q_1, \ldots, q_{n-1}, W$ where $a \in \Sigma$, $q, q_0, \ldots, q_{n-1} \in Q$, and $R, W \in \{\{r\} \mid r \in \mathcal{R}\} \cup \{\emptyset\}$.

The automaton traverses labelled trees where the set of labels is Σ. As before, we assume that all the rules respect the arity so that $\mathsf{arity}(a) = n$ in the definition above.

The operational semantics for such automaton is as follows. Configurations of $\mathcal{A}$ in a tree t are elements of $\mathsf{Config} = \mathsf{dom}(t) \times Q \times \mathsf{Reg_b}$ where $\mathsf{Reg_b} = \mathcal{R} \to \{0, 1\}$. Suppose we are in a configuration $\langle \pi, q, f \rangle$. Consider a rule

$$a, q, R \rightsquigarrow q_0, q_1, \ldots, q_{n-1}, W$$

the rule is *applicable* when

- $R = \emptyset$ and $t(\pi) = a$, or
- $R = \{r\}$ for some $r \in \mathcal{R}$, $t(\pi) = a$, and $f(r) = 1$.

When the rule is applied, the automaton forks the computation to all n sons of the tree (note that the arity must be respected both by the tree and by the rule) and for a node $i \in \overline{n}$ the *i-th resulting configuration* is $\langle \pi \cdot i, q_i, f^W_\Downarrow \rangle$ where $f^W_\Downarrow : \mathcal{R} \to \{0, 1\}$ is defined as

$$f^W_\Downarrow(l) = \begin{cases} f(l) & \text{for } l \notin W, \\ 1 & \text{for } l \in W. \end{cases} \tag{6}$$

Note that in case $W = \emptyset$ the condition in the second case of the definition is not possible so this pattern defines $f^W_\Downarrow$ equal to f. Again, we drop W whenever it is clear from the context.

A *run* of an automaton $\mathcal{A}$ on a tree t is a function $\mathfrak{r} : \mathsf{dom}(t) \to \mathsf{Config}$ that respects the rules of δ, i.e. for each node $\pi \in \mathsf{dom}(t)$ there is a rule

$$a, q, R \rightsquigarrow q_0, q_1, \ldots, q_{n-1}, W \in \delta$$

that is applicable in the configuration $\mathfrak{r}(\pi)$ and for each son i of the node π the configuration $\mathfrak{r}(\pi \cdot i)$ is the i-th resulting configuration of the rule.

We say that an automaton $\mathcal{A}$ *accepts a tree t from a configuration* $\langle \pi, q, f \rangle$ when there is a correct run of $\mathcal{A}$ on $t|_\pi$ that starts with the configuration $\langle \varepsilon, q, f \rangle$. Let us define the function $f_I : \mathcal{R} \to \{0, 1\}$ so that $f_I(r) = 0$ for $r \in \mathcal{R}$. We say that the automaton $\mathcal{A}$ *accepts a tree t* when there is a correct run of the automaton that starts in $\langle \varepsilon, q_I, f_I \rangle$. The set of all trees that $\mathcal{A}$ accepts is written $L(\mathcal{A})$.

We define a translation of one operation IM's to binary automata. Given a one operation inhabitation machine $\mathcal{A} = \langle \Sigma, N, Q, q_I, \mathcal{R}, \delta \rangle$ we fix a set $N_0 \subseteq N$ of size equal to $|\mathcal{R}|$ together with a bijection from $\mathcal{R}$ to N_0. We write n_r for the result of the bijection on an element $r \in \mathcal{R}$. We define a binary automaton $\mathcal{B} = \langle \Sigma_\mathcal{B}, Q_\mathcal{B}, q_{\mathcal{B},I}, \mathcal{R}_\mathcal{B}, \delta_\mathcal{B} \rangle$ where $\Sigma_\mathcal{B} = \Sigma \cup \Sigma \times N_0$, $Q_\mathcal{B} = Q$, $q_{\mathcal{B},I} = q_I$, $\mathcal{R}_\mathcal{B} = \mathcal{R}$, $\delta_\mathcal{B}$ contains for each rule

$$a, q, R \rightsquigarrow q_0, q_1, \ldots, q_{n-1}, W \in \delta \qquad \text{the rule} \qquad a', q, R \rightsquigarrow q_0, q_1, \ldots, q_{n-1}, W$$

where

- $a' = a$ when $R = W = \emptyset$,
- $a' = \langle a, n_r \rangle$ when $R = \emptyset$ and $W = \{r\}$,
- $a' = \langle a, n_r \rangle$ when $R = \{r\}$ and $W = \emptyset$.

We can now define the transformation operations for registers.

▶ **Definition 18** (register transformations). For $f : \mathcal{R} \to P_{\mathsf{fin}}(N)$ we define $f^\bullet : \mathcal{R} \to \{0, 1\}$ as $f^\bullet(r) = 0$ when $f(r) = \emptyset$ and $f^\bullet(r) = 1$ otherwise.

For $f : \mathcal{R} \to \{0, 1\}$ we define $f_\bullet : \mathcal{R} \to P_{\mathsf{fin}}(N)$ as $f_\bullet(r) = \emptyset$ when $f(r) = 0$ and $f_\bullet(r) = \{n_r\}$ where $n_r \in N_0$ otherwise.

▶ **Proposition 19.**

1. *If $\mathcal{A}$ accepts a tree t from a configuration $\langle \varepsilon, q, f \rangle$ then there is some t' such that $\mathcal{B}$ accepts t' from the configuration $\langle \varepsilon, q, f^\bullet \rangle$.*
2. *If $\mathcal{B}$ accepts a tree t from a configuration $\langle \varepsilon, f, q \rangle$ then $\mathcal{A}$ accepts t from the configuration $\langle \varepsilon, q, f_\bullet \rangle$.*

Proof. Both the proof of (1) and the proof of (2) are by induction over t. We only sketch the proof due to the lack of space. For illustration we present here a fragment of the proof for (1). The subcase concerns an internal node of the tree t. We know that a rule of the form

$$a, q, R \rightsquigarrow q_0, q_1, \ldots, q_{n-1}, W \in \delta$$

was applied in the configuration $\langle \varepsilon, q, f \rangle$ to accept t. We consider the subcase when $R = \emptyset$, $W = \{r\}$ and $t(\varepsilon) = \langle a, x \rangle$. We can consider the trees $t|_i$ and configurations $\langle \varepsilon, q_i, f_\downarrow \rangle$ for $i \in \overline{n}$ where $f_\downarrow$ is defined as in the pattern (2) on page 133. The automaton $\mathcal{A}$ accepts these trees from respective configurations. By the induction hypothesis there are trees t'_i for $i \in \overline{n}$ that $\mathcal{B}$ accepts from the configurations $\langle \varepsilon, q_i, (f_\downarrow)^\bullet \rangle$ for $i \in \overline{n}$ respectively. Note now that $(f_\downarrow)^\bullet = (f^\bullet)_\Downarrow$ so actually $\mathcal{B}$ accepts the trees from the configurations $\langle \varepsilon, q_i, (f^\bullet)_\Downarrow \rangle$. We can now define the tree $t' = \langle a, n_r \rangle (t'_0, \ldots, t'_{n-1})$ and by the definition of $\mathcal{B}$ the automaton contains the rule $a, q, \emptyset \rightsquigarrow q_0, \ldots, q_{n-1}, r \in \delta_{\mathcal{B}}$. This rule is applicable in the configuration $\langle \varepsilon, q, f^\bullet \rangle$ and leads to the mentioned above acceptable configurations $\langle \varepsilon, q_i, (f^\bullet)_\Downarrow \rangle$, which guarantees that $\mathcal{B}$ accepts t'. ◀

▶ **Proposition 20.** *The emptiness problem for one operation IM's is in PSPACE.*

Proof. To certify that the emptiness problem is in PSPACE we give a polynomial time alternating algorithm that given a machine $\mathcal{A} = \langle \Sigma, N, Q, q_I, \mathcal{R}, \delta \rangle$ checks for emptiness of $L(\mathcal{A})$. We first transform $\mathcal{A}$ to its corresponding binary automaton $\mathcal{B}$. Next, the algorithm keeps in its memory the configuration of $\mathcal{B}$ and a counter i of the number of steps. The initial configuration is $\langle \varepsilon, q_I, f \rangle$ and the initial counter $i = 0$. The algorithm proceeds by executing in loop the following three steps

1. it non-deterministically chooses a transition $a, q, R \rightsquigarrow q_0, \ldots, q_{n-1}, W \in \delta_{\mathcal{B}}$ and then
2. it universally moves to n configurations that result from applying the rule and that have $q_1, \ldots, q_n$ in their coordinates,
3. it increments i and in case $i > i_{\mathsf{max}} = k \cdot |Q|$ it leaves the loop with failure.

Observe that the loop is finished not only in step (3), but also in step (1) when a rule is chosen so that there are no states on the right-hand side of the chosen rule. In case the algorithm leaves the loop in this way it accepts. The bound $k \cdot |Q|$ is chosen so that in case there are more steps the state with register content must repeat (note that once a register is set to 1 it cannot be turned back to 0).

In this way the algorithm creates a potential tree t along a correct run on it. A closer examination of the procedure shows that this tree is in total discharge form. ◀

▶ **Proposition 21.** *The emptiness problem for IM's is hard for PSPACE.*

Proof. In Theorem 10, we defined machines that express inhabitation in $\lambda_{\rightarrow}$, which is PSPACE-hard [16]. ◀

As an immediate corollary of Proposition 20 and 21 we obtain the following theorem.

▶ **Theorem 22.** *The emptiness problem for one operation IM's is PSPACE-complete.*

5 Conclusions and further work

We have presented a model of automata and discussed how it corresponds to the inhabitation problem for the simply typed λ-calculus. As this was done for syntax with named binders, it is interesting to see how this would look like with de Bruijn indices. The binary automata presented in Section 4 recognise the language of terms in total discharge form. It would be interesting to see their version for depth bounded calculus of Dyckhoff-Hudelmeier [5, 10].

In addition to the presented closure properties for sum and intersection of languages one traditionally considers other operations such as substitution of languages, or cilindrification. The question concerning the closure for the operations remains open. Another interesting direction of study would be to give automata that deal with full propositional intuitionistic logic (i.e. one that includes logical alternative, conjunction, and negation). Automata for infinite tree languages with Büchi acceptance conditions can give a similar account to the one obtained in our Theorem 10 for Böhm trees.

We believe that Theorem 14 concerning the invariance of α-conversion can be generalised to a wider class of binding operators and to α-conversion that is expressed as a permutation of variables. In this way we would effectively obtain automata adequate for the Gabbay and Pitts [7] approach to binder syntax.

One more interesting direction would be to augment our automata with additional primitives that make it possible to recognise expressions in the relation of β-reduction. We believe that the automata with global equality constraints [2] can give here promising results.

References

1 Hendrik Pieter Barendregt, Wil Dekkers, and Richard Statman. *Lambda Calculus with Types*. Perspectives in logic. Cambridge University Press, 2013.

2 Luis Barguñó, Carles Creus, Guillem Godoy, Florent Jacquemard, and Camille Vacher. The emptiness problem for tree automata with global constraints. In Jean-Pierre Jouannaud, editor, *Proceedings of the LICS 2010*, pages 263–272. IEEE Computer Society, 2010.

3 Choukri-Bey Ben-Yelles. *Type-assignment in the lambda-calculus; syntax and semantics*. PhD thesis, Mathematics Department, University of Wales, Swansea, UK, 1979.

4 Sabine Broda and Luís Damas. On long normal inhabitants of a type. *Journal of Logic and Computation*, 15(3):353–390, 2005.

5 Roy Dyckhoff. Contraction-free sequent calculi for intuitionistic logic. *Journal of Symbolic Logic*, 57:795–807, 9 1992.

6 Boris Düdder, Moritz Martens, and Jakob Rehof. Staged composition synthesis. In Zhong Shao, editor, *Proceedings of ESOP 2014*, volume 8410 of *LNCS*, pages 67–86. Springer, 2014.

7 Murdoch J. Gabbay and Andrew M. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects of Computing*, 13(3-5):341–363, 2002.

8 Silvia Ghilezan. Inhabitation in intersection and union type assignment systems. *Journal of Logic and Computation*, 3(6):671–685, 1993.

9 J. Roger Hindley. *Basic simple type theory*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, New York, 1996.

10 Jörg Hudelmaier. An o(n log n)-space decision procedure for intuitionistic propositional logic. *Journal of Logic and Computation*, 3(1):63–75, 1993.

11 Michael Kaminski and Nissim Francez. Finite-memory automata. *Theoretical Computer Science*, 134(2):329–363, 1994.

12 Michael Kaminski and Tony Tan. Tree automata over infinite alphabets. In A. Avron, N. Dershowitz, and A. Rabinovich, editors, *Pillars of Computer Science, Essays Dedicated to Boris (Boaz) Trakhtenbrot on the Occasion of His 85th Birthday*, volume 4800 of *LNCS*, pages 386–423. Springer, 2008.

13 C.-H. Luke Ong and Nikos Tzevelekos. Functional reachability. In *Proceedings of the 24th Annual IEEE Symposium on Logic in Computer Science, LICS 2009, 11-14 August 2009, Los Angeles, CA, USA*. IEEE Computer Society, 2009.

14 Dag Prawitz. *Natural Deduction*. Almqvist and Wiksell, Sweden, 1965.

15 Sylvain Salvati and Igor Walukiewicz. Krivine machines and higher-order schemes. *Information and Computation*, 239:340–355, 2014.

16 Richard Statman. Intuitionistic propositional logic is PSPACE-complete. *Theoretical Computer Science*, 9(1):67–72, 1979.

17 Colin Stirling. Dependency tree automata. In Luca de Alfaro, editor, *Proceedings of FOSSACS 2009*, volume 5504 of *LNCS*, pages 92–106. Springer, 2009.

18 Masako Takahashi, Yohji Akama, and Sachio Hirokawa. Normal proofs and their grammar. *Information and Computation*, 125(2):144–153, 1996.

19 Anne Troelstra and Helmut Schwichtenberg. *Basic Proof Theory*. Cambridge University Press, 1996, 2000.

20 Paweł Urzyczyn. Inhabitation of low-rank intersection types. In Pierre-Louis Curien, editor, *Proceedings of TLCA 2009*, volume 5608 of *LNCS*, pages 356–370. Springer, 2009.

21 Mordchaj Wajsberg. Untersuchungen über den Aussagenkalkül von A. Heyting. *Wiadomości Matematyczne*, 46, 1938. English translation: On A. Heyting's propositional calculus, in *Mordchaj Wajsberg, Logical Works* (S. J. Surma, editor), Ossolineum, Wrocław, 1977, pages 132–171.

Maximal Partition Logic: Towards a Logical Characterization of Copyless Cost Register Automata

Filip Mazowiecki[1] and Cristian Riveros[2]

1 **University of Warsaw, Poland**
2 **Pontificia Universidad Católica de Chile, Chile**

Abstract

It is highly desirable for a computational model to have a logic characterization like in the seminal work of Büchi that connects MSO with finite automata. For example, weighted automata are the quantitative extension of finite automata for computing functions over words and they can be naturally characterized by a subfragment of weighted logic introduced by Droste and Gastin. Recently, cost register automata (CRA) were introduced by Alur et al. as an alternative model for weighted automata. In hope of finding decidable subclasses of weighted automata, they proposed to restrict their model with the so-called copyless restriction. Unfortunately, copyless CRA do not enjoy good closure properties and, therefore, a logical characterization of this class seems to be unlikely.

In this paper, we introduce a new logic called maximal partition logic (MP) for studying the expressiveness of copyless CRA. In contrast to the previous approaches (i.e. weighted logics), MP is based on a new set of "regular" quantifiers that partition a word into maximal subwords, compute the output of a subformula over each subword separately, and then aggregate these outputs with a semiring operation. We study the expressiveness of MP and compare it with weighted logics. Furthermore, we show that MP is equally expressive to a natural subclass of copyless CRA. This shows the first logical characterization of copyless CRA and it gives a better understanding of the copyless restriction in weighted automata.

1998 ACM Subject Classification F.4.1. Computational logic

Keywords and phrases MSO, Finite Automata, Cost Register Automata, Weighted Automata, Weighted Logics, Semirings

Digital Object Identifier 10.4230/LIPIcs.CSL.2015.144

1 Introduction

Weighted automata are an extension of finite state automata to compute functions over strings [8]. They have been extensively studied since Schützenberger [21], and its decidability problems [15, 2], extensions [7], and applications [18, 6] have been deeply investigated. From the logic-side, Weighted MSO logic (WMSO) has been introduced and investigated in [7, 14]. This logic is a quantitative extension of MSO to define functions over strings and its natural fragment gives a logic-based characterization of weighted automata.

Recently, Alur et al. [3] introduced the computational model of cost register automata (CRA), an alternative model to weighted automata for computing functions. The main idea of this model is to enhance deterministic finite automata with registers that can be combined with semiring operations, but the registers cannot be used for taking decisions during a computation. Alur et al. show in [3] that a fragment of CRA is equally expressive to weighted automata, but the general model is strictly more expressive.

24th EACSL Annual Conference on Computer Science Logic (CSL 2015).
Editor: Stephan Kreutzer; pp. 144–159

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The main advantage of introducing a new model is that it allows to study natural subclasses of functions that do not arise naturally in the classical framework. This is the case for the class of copyless CRA that where proposed in [3]. The idea of the so-called copyless restriction is to use each register at most once in every transition. Intuitively, the automaton model is register-deterministic in the sense that it cannot copy the content of each register, similar to a deterministic finite automaton that cannot make a copy of its current state. Copyless CRA is also an excellent candidate for having good decidability properties. It was stated in [3] that the existing proofs of undecidability in weighted automata rely on the unrestricted non-deterministic nature of the model and, thus, it might be possible that copyless CRA can have good decidability properties [3]. Despite that this is a natural and interesting model for computing functions, research on this line has not been pursued further and not much is known about copyless CRA.

In this paper, we introduce a new logic called *Maximal Partition Logic* (MP) to define functions over strings. In contrast to the previous approaches (WMSO), MP is based on a different set of quantifiers and it does not need to distinguish between a boolean or quantitative level of evaluation (see [7, 14]). MP is based on regular quantifiers that partition a string into maximal substrings, compute a subformula over each substring separately and then aggregate these outputs with respect to a semiring operation. Recently in [5] a logic with a similar flavor has been proposed but in a different context, namely for data words. The authors define a syntactically restricted fragment of MSO formulas with two free variables called rigid MSO-formulas. Each assignment of the free variables can be seen as choosing the substrings between the assigned positions. The rigid formulas put restrictions in the chosen set of substrings that coincides with our restriction of choosing maximal substrings.

WMSO has the drawbacks of its automata counterpart (weighted automata) – the lack of good decidability properties [2, 7, 14, 15]. We show that MP is less expressive than WMSO and even less expressive than weighted automata. Interestingly, MP can still define natural functions and it is strictly more expressive than finitely ambiguous weighted automata, a subclass of weighted automata, which has good decidability properties. In this paper we study the expressiveness of MP and compare its expressiveness with WMSO and fragments of WMSO. By this comparison, MP might be a good candidate for a logic with good decidability properties.

The main result of this paper is that MP is equally expressive to a natural fragment of copyless CRA, called *bounded alternation copyless* CRA (BAC). This fragment of copyless CRA has good closure properties and, at the same time, it does not lose much in terms of expressibility. Most examples in [3] and this paper are definable by BAC automata. This result could also be the first step in proving the decidability of MP. For example a positive answer to a decidability problem for copyless CRA will imply a positive answer for the same decidability problem for MP.

Organization. In Section 2 we introduce CRA and some basic definitions. In Section 3 we introduce MP and compare it with other formalisms. In particular we discuss the connection between this logic and rigid formulas. In Section 4 we define BAC automata and prove that that this class of automata is equally expressive to MP. In Section 5 we compare the expressiveness of MP with WMSO. We conclude in Section 6 with possible directions for future research. Due to the page limit some proofs are moved to the appendix, available online.

2 Preliminaries

In this section, we summarize the notation and definitions used for finite automata, regular expressions, MSO logic and cost register automata.

Finite automata over strings. Let Σ be a finite set of symbols. We denote by Σ^* the set of all finite strings over Σ and by ϵ the empty string in Σ^*. The length of a string $w \in \Sigma^*$ is denoted by $|w|$. Furthermore, for any $a \in \Sigma$ the number of a-symbols in w is denoted by $|w|_a$.

A finite automaton [11] over Σ^* is a tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ where Q is a finite set of states, $\delta \subseteq Q \times \Sigma \times Q$ is a finite transition relation, q_0 is the initial state and F is the set of final states. A run ρ of $\mathcal{A}$ is a sequence of transitions of the form: $p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} p_n$ where $(p_i, a_{i+1}, p_{i+1}) \in \delta$ for every $i < n$. We say that ρ (like above) is a run of $\mathcal{A}$ over $w = a_1 \dots a_n$ if $p_0 = q_0$. Furthermore, we say that ρ is an accepting run if $p_n \in F$. A string w is accepted by $\mathcal{A}$ if there exists an accepting run of $\mathcal{A}$ over w. We denote by $\mathcal{L}(\mathcal{A})$ the language of all strings accepted by $\mathcal{A}$. A finite automaton $\mathcal{A}$ is called deterministic if δ is a function of the form $\delta : Q \times \Sigma \to Q$.

Regular expressions. Let Σ be an alphabet. The syntax of regular expressions [11] over Σ is given by:

$$R := \varnothing \mid \epsilon \mid a \mid R \cdot R \mid R + R \mid R^*$$

where $a \in \Sigma$. The semantics of regular expressions over strings is defined as usual [11]. We write $\mathcal{L}(R)$ to denote the set of all strings that satisfy the regular expression R.

MSO. Let Σ be an alphabet. The syntax of an MSO-formula over Σ-strings is given by:

$$\varphi := P_a(x) \mid x \leq y \mid x \in X \mid (\varphi \vee \varphi) \mid \neg\varphi \mid \exists x.\, \varphi \mid \exists X.\, \varphi$$

where $a \in \Sigma$, x and y are first-order variables and X is a set of variables. Let $w = a_1 \dots a_n \in \Sigma^*$ be a string. We represent the string w as a structure $(\{1, \dots, n\}, \leq, (P_a)_{a \in \Sigma})$, where $P_a = \{i \mid a_i = a\}$. Further, we denote by $\mathrm{dom}(w) = \{1, \dots, n\}$ the domain of w as a structure. Given a finite set $\bar{x}$ of first-order and second-order variables, an $(\bar{x}, w)$-assignment σ is a function that maps every first order variable in $\bar{x}$ to $\mathrm{dom}(w)$ and every second order variable in $\bar{x}$ to $2^{\mathrm{dom}(w)}$. Furthermore, we denote by $\sigma[x \to i]$ the extension of the $(\bar{x}, w)$-assignment σ such that $\sigma[x \to i](x) = i$ and $\sigma[x \to i](y) = \sigma(y)$ for all variables $y \neq x$. Consider an MSO-formula $\varphi(\bar{x})$ and a $(\bar{x}, w)$-assignment σ. We write $w \vDash \varphi(\sigma)$ if (w, σ) satisfies $\varphi(\bar{x})$ using the standard MSO-semantics.

Semirings and functions. A semiring is a structure $\mathbb{S} = (S, \oplus, \odot, \mathbb{0}, \mathbb{1})$ where $(S, \oplus, \mathbb{0})$ is a commutative monoid, $(S - \mathbb{0}, \odot, \mathbb{1})$ is a monoid, multiplication distributes over addition, and $\mathbb{0} \odot s = s \odot \mathbb{0} = \mathbb{0}$ for each $s \in S$. If the multiplication is commutative, we say that $\mathbb{S}$ is commutative. In this paper, we always assume that $\mathbb{S}$ is commutative. For the sake of simplicity, we usually denote the set of elements S by the name of the semiring $\mathbb{S}$. As standard examples of semirings we will consider the *semiring of natural numbers* $\mathbb{N}(+, \cdot) = (\mathbb{N}, +, \cdot, 0, 1)$, the *min-plus semiring* $\mathbb{N}_\infty(\min, +) = (\mathbb{N}_\infty, \min, +, \infty, 0)$ and the *max-plus semiring* $\mathbb{N}_{-\infty}(\max, +) = (\mathbb{N}_{-\infty}, \max, +, -\infty, 0)$ which are standard semirings in the field of weighted automata [8].

In this paper, we study the specification of functions from strings to values, namely, from Σ^* to $\mathbb{S}$. We say that a function $f : \Sigma^* \to \mathbb{S}$ is definable by a computational system $\mathcal{A}$ (e.g.

weighted automaton, or CRA) if $f(w) = [\![\mathcal{A}]\!](w)$ for any $w \in \Sigma^*$ where $[\![\mathcal{A}]\!]$ is the semantics of $\mathcal{A}$ over strings. For any string w, we denote by w^r the reverse string. We say that a class of functions F is *closed under reverse* [3] if for every $f \in F$ there exists a function $f^r \in F$ such that $f^r(w) = f(w^r)$ for all $w \in \Sigma^*$.

Variables, expressions, and substitutions. Fix a semiring $\mathbb{S} = (S, \oplus, \odot, \mathbb{0}, \mathbb{1})$ and a set of variables $\mathcal{X}$ disjoint from S. We denote by $\mathrm{Expr}(\mathcal{X})$ the set of all syntactical *expressions* that can be defined from $\mathcal{X}$, constants in S, and the syntactical signature of $\mathbb{S}$. For any expression $e \in \mathrm{Expr}(\mathcal{X})$ we denote by $\mathrm{Var}(e)$ the set of variables in e. We call an expression $e \in \mathrm{Expr}(\mathcal{X})$ without variables (i.e. $\mathrm{Var}(e) = \varnothing$) a *ground* expression. For any ground expression we define $[\![e]\!] \in \mathbb{S}$ to be the evaluation of e with respect to $\mathbb{S}$.

A *substitution* over $\mathcal{X}$ is defined as a mapping $\sigma : \mathcal{X} \to \mathrm{Expr}(\mathcal{X})$. We denote the set of all substitutions over $\mathcal{X}$ by $\mathrm{Subs}(\mathcal{X})$. A *ground substitution* σ is a substitution where each expression $\sigma(x)$ is ground for each $x \in \mathcal{X}$. Any substitution σ can be extended to a mapping $\hat{\sigma} : \mathrm{Expr}(\mathcal{X}) \to \mathrm{Expr}(\mathcal{X})$ such that, for every $e \in \mathrm{Expr}(\mathcal{X})$, $\hat{\sigma}(e)$ is the resulting expression $e[\sigma]$ of substituting each $x \in \mathrm{Var}(e)$ by the expression $\sigma(x)$. For example, if $\sigma(x) = 2x$ and $\sigma(y) = 3y$, and $e = x + y$, then $\hat{\sigma}(e) = 2x + 3y$. By using the extension $\hat{\sigma}$, we can define the composition substitution $\sigma_1 \circ \sigma_2$ of two substitutions σ_1 and σ_2 such that $\sigma_1 \circ \sigma_2(x) = \hat{\sigma}_1(\sigma_2(x))$ for each $x \in \mathcal{X}$.

A valuation is defined as a substitution of the form $\nu : \mathcal{X} \to \mathbb{S}$. We denote the set of all valuations over $\mathcal{X}$ by $\mathrm{Val}(\mathcal{X})$. Clearly, any valuation ν composed with a substitution σ defines an expression without variables that can be evaluated as $[\![\nu \circ \sigma(x)]\!]$ for any $x \in \mathcal{X}$.

In this paper, we say that two expressions e_1 and e_2 are equal (denoted by $e_1 = e_2$) if they are equal up to evaluation equivalence, that is, $[\![\hat{\nu}(e_1)]\!] = [\![\hat{\nu}(e_2)]\!]$ for every valuation $\nu \in \mathrm{Val}(\mathcal{X})$. Similarly, we say that two substitutions σ_1 and σ_2 are equal (denoted by $\sigma_1 = \sigma_2$) if $\sigma_1(x) = \sigma_2(x)$ for every $x \in \mathcal{X}$.

Cost register automata. A cost register automaton (CRA) over a semiring $\mathbb{S}$ [3] is a tuple $\mathcal{A} = (Q, \Sigma, \mathcal{X}, \delta, q_0, \nu_0, \mu)$ where Q is a set of states, Σ is the input alphabet, $\mathcal{X}$ is a set of variables (we also call them registers), $\delta : Q \times \Sigma \to Q \times \mathrm{Subs}(\mathcal{X})$ is the transition function, q_0 is the initial state, $\nu_0 : \mathcal{X} \to \mathbb{S}$ is the initial valuation, and $\mu : Q \to \mathrm{Expr}(\mathcal{X})$ is the final output function. A configuration of $\mathcal{A}$ is a tuple (q, ν) where $q \in Q$ and $\nu \in \mathrm{Val}(\mathcal{X})$ represents the current values in the variables of $\mathcal{A}$. Given a string $w = a_1 \dots a_n \in \Sigma^*$, the run of $\mathcal{A}$ over w is a sequence of configurations: $(q_0, \nu_0) \xrightarrow{a_1} (q_1, \nu_1) \xrightarrow{a_2} \dots \xrightarrow{a_n} (q_n, \nu_n)$ such that, for every $1 \le i \le n$, $\delta(q_{i-1}, a_i) = (q_i, \sigma_i)$ and $\nu_i(x) = [\![\nu_{i-1} \circ \sigma_i(x)]\!]$ for each $x \in \mathcal{X}$. The output of $\mathcal{A}$ over w, denoted by $[\![\mathcal{A}]\!](w)$, is $[\![\hat{\nu}_n(\mu(q_n))]\!]$.

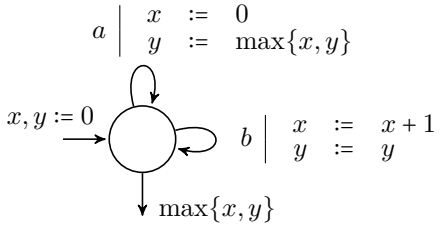
The run of $\mathcal{A}$ over w can be equally defined in terms of ground expressions rather than values. A ground configuration of $\mathcal{A}$ is a tuple (q, ς) where $q \in Q$ and $\varsigma \in \mathrm{Subs}(\mathcal{X})$ is a ground substitution. Given a string $w = a_1 \dots a_n \in \Sigma^*$, the ground run of $\mathcal{A}$ over w is a sequence of ground configurations: $(q_0, \varsigma_0) \xrightarrow{a_1} \dots \xrightarrow{a_n} (q_n, \varsigma_n)$ such that for $1 \le i \le n$, $\delta(q_{i-1}, a_i) = (q_i, \sigma_i)$, $\varsigma_0 = \nu_0$ and $\varsigma_i(x) = \hat{\varsigma}_{i-1}\big(\sigma_i(x)\big)$ for each $x \in \mathcal{X}$. We denote the output ground expression of $\mathcal{A}$ over a string w by $|\mathcal{A}|(w) = \hat{\varsigma}_n(\mu(q_n))$. Notice that, in contrast to ordinary runs, ground runs keep ground expressions as partial values of the run. It is easy to see that $[\![\mathcal{A}]\!](w) = [\![|\mathcal{A}|(w)]\!]$.

Copyless restriction and copyless CRA. We say that an expression $e \in \mathrm{Expr}(\mathcal{X})$ is *copyless* if e uses every variable from $\mathcal{X}$ at most once. For example, $x \cdot (y + z)$ is copyless but $x \cdot y + x \cdot z$ is not copyless (because x is mentioned twice). Notice that the copyless restriction is a syntactical constraint over expressions. Furthermore, we say that a substitution σ is *copyless*

if for every $x \in \mathcal{X}$ the expression $\sigma(x)$ is copyless and $\mathrm{Var}(\sigma(x)) \cap \mathrm{Var}(\sigma(y)) = \varnothing$ for every pair of different registers $x, y \in \mathcal{X}$. Copyless substitutions, similar to copyless expressions, are restricted in such a way that each variable is used at most once in the whole substitution.

A CRA $\mathcal{A}$ is called *copyless* if for every transition $\delta(q_1, a) = (q_2, \sigma)$ the substitution σ is copyless; and for every state $q \in Q$ the expression $\mu(q)$ is copyless, where μ is the output function of $\mathcal{A}$. In other words, every time that registers from $\mathcal{A}$ are operated, they can be used just once. In the following, we give some examples of copyless CRA.

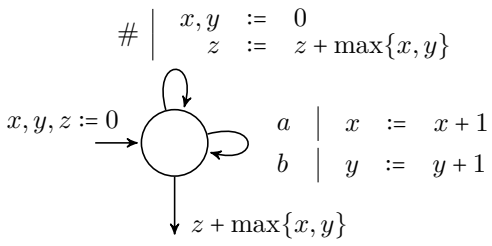
▶ **Example 1.** Let $\mathbb{S}$ be the max-plus semiring $\mathbb{N}_{-\infty}(\max, +)$ and $\Sigma = \{a, b\}$. Consider the function f_1 that for a given string $w \in \Sigma^*$ computes the longest substring of b's. This can be easily defined by the following CRA $\mathcal{A}_1$ with two registers x and y.



$\mathcal{A}_1$ stores in the x-register the length of the last suffix of b's and in the y-register the length of the longest substring of b's seen so far. After reading a b-symbol $\mathcal{A}_1$ adds one to x (the b-infix has increased by one) and it keeps y unchanged. Furthermore, after reading an a-symbol it resets x to zero and updates y by comparing the substring of b's that has just finished (i.e. the previous x-content) with the length of the longest substring of b's (i.e. the previous y-content) that has been seen so far. Finally, it outputs the maximum between x and y.

One can easily check that the previous CRA satisfies the copyless restriction and, therefore, it is a copyless CRA. Indeed, each substitution is copyless and the final output expression $\max\{x, y\}$ is copyless as well.

▶ **Example 2.** Again, let $\mathbb{S}$ be the max-plus semiring $\mathbb{N}_{-\infty}(\max, +)$ and $\Sigma = \{a, b, \#\}$. Consider the function f_2 such that, for any $w \in \Sigma^*$ of the form $w_0 \# w_1 \# \dots \# w_n$ with $w_i \in \{a, b\}^*$, it computes the maximum number of a's or b's for each substring w_i (i.e. $\max\{|w_i|_a, |w_i|_b\}$) and then it sums these values over all substrings w_i, that is, $f_2(w) = \sum_{i=0}^{n} \max\{|w_i|_a, |w_i|_b\}$. One can check that the copyless CRA $\mathcal{A}_2$ defined below computes f_2:



In the above diagram of $\mathcal{A}_2$, we omit an assignment if a register is not updated (i.e. it keeps its previous value). For example, for the a-transition we omit the assignments $y := y$ and $z := z$ for the sake of presentation of the CRA. Similarly, we also omit the assignment $x := x$ and $z := z$ for the b-transition. One should keep in mind these assignments because of the copyless restriction.

The copyless CRA $\mathcal{A}_2$ follows similar ideas to $\mathcal{A}_1$: the registers x and y count the number of a's and b's, respectively, in the longest suffix without $\#$ and the register z stores the

partial output without considering the last suffix of a's and b's. When the last substring w_i over $\{a, b\}$ is finished (i.e. there comes a #-symbol or the input ends), then $\mathcal{A}_2$ adds the maximum number of a's or b's in w_i to z (i.e. $z := z + \max\{x, y\}$).

Trim assumption. For technical reasons, in this paper we assume that our finite automata and cost register automata are always *trim*, namely, all their states are reachable from some initial states (i.e., they are accessible) and they can reach some final states (i.e., they are co-accessible). It is worth noticing that verifying if a state is accessible or co-accessible is reduced to a reachability test in the transition graph [19]; and this can be done in NLogSpace. Thus, we can assume without lost of generality that all our automata are trimmed.

3 A quantitative logic based on partitions

3.1 Regular selectors

In this subsection we extend regular expressions for selecting intervals from a string. Our approach is similar to the one in [9, 12], but we restrict the selection to just a set of intervals (i.e. spans in [9]) instead of relations of intervals.

Fix a string $w \in \Sigma^*$. An interval of w is a pair (i, j) such that $1 \leq i \leq j \leq |w|$. We write $\text{Int}(w)$ for the set of all intervals of w. For an interval (i, j), we denote by $w[i, j]$ the substring between positions i and j, by $w[\cdot, j]$ the prefix of w until position j and by $w[i, \cdot]$ the suffix of w starting from position i. For the sake of simplification, we define $w[\cdot, i]$ and $w[i, \cdot]$ equal to ϵ whenever $i \notin \{1, \ldots, |w|\}$.

A *regular selector* (RS) over Σ (or just selector or triple) is a triple (R, S, T) where R, S, and T are regular expressions over Σ. The set of all selectors over Σ is denoted by RS_Σ. We usually write $R\langle S\rangle T$ instead of (R, S, T). The main motivation of a selector (R, S, T) is to select intervals (i, j) from a string w by dividing w into $w = xyz$ such that x, y, and z match R, S, and T, respectively, and $w[i, j] = y$. Specifically, we say that an interval (i, j) of a string w is selected by a triple $R\langle S\rangle T$ if, and only if, $w[\cdot, i-1] \in \mathcal{L}(R)$, $w[i, j] \in \mathcal{L}(S)$, and $w[j+1, \cdot] \in \mathcal{L}(T)$. The set of all intervals of w selected by $R\langle S\rangle T$ is defined as:

$$\text{Sel}(w, R\langle S\rangle T) \;=\; \left\{ (i, j) \in \text{Int}(w) \;\mid\; w[\cdot, i-1] \in \mathcal{L}(R) \wedge w[i, j] \in \mathcal{L}(S) \wedge w[j+1, \cdot] \in \mathcal{L}(T) \right\}$$

▶ **Example 3.** Let $\Sigma = \{a, b\}$. Suppose that we want to define all maximal intervals that define substrings of b-symbols in a string. This can be defined by the following regular selector:

$$((a + b)^* a + \epsilon)\ \langle b^+ \rangle\ (a(a + b)^* + \epsilon)$$

The purpose of a selector $R\langle S\rangle T$ is to extract all intervals that satisfy the regular expression S under the context defined by R and T. In our logic, we restrict the semantics of selectors to consider just intervals that are maximal in terms of containment. More precisely, we say that an interval (i_1, j_1) is contained in an interval (i_2, j_2) (denoted by $(i_1, j_1) \sqsubseteq (i_2, j_2)$) if, and only if, $i_2 \leq i_1$ and $j_1 \leq j_2$. The $\sqsubseteq$-relation basically defines a partial order between intervals and we can talk about the $\sqsubseteq$-maximal intervals of a set. We write $\text{Max}_\sqsubseteq(I)$ to denote the set of all maximal intervals in I with respect to the partial order $\sqsubseteq$ for any set I of intervals. Given a selector $\mathtt{R} = R\langle S\rangle T$ and a string w, we define the set of intervals selected by $\mathtt{R}$ over w under maximal semantics by:

$$\text{Max}(w, R\langle S\rangle T) \;=\; \text{Max}_\sqsubseteq(\text{Sel}(w, R\langle S\rangle T))$$

That is, under the maximal semantics we select just intervals that are maximal with respect to the partial order $\sqsubseteq$. This new semantics simplifies selectors from Example 3.

▶ **Example 4.** With the maximal semantics, we can easily define the the set of maximal intervals that define substrings of b-symbols like in Example 3. By using the maximal semantics we can define this set of intervals easily as follows:

$$(a+b)^* \ \langle b^+ \rangle \ (a+b)^*$$

We usually do not need the context R and T when we are using the maximal semantics. For instance, in the previous example R and S were equal to $(a+b)^*$ and could be omitted. For the sake of simplification, we usually omit R, T and the angular brackets whenever R and T are both equivalent to Σ^*. We can simplify the above selector and just write b^+ to select the maximal intervals of b's.

3.2 Maximal partition logic

For a fixed semiring $\mathbb{S} = (S, \oplus, \odot, \mathbb{0}, \mathbb{1})$ and an alphabet Σ we define the *maximal partition logic* (MP). This is a logic for computing functions similar to weighted logics [7] but with a different set of quantifiers that are parametrized by regular selectors. Formally, the formulas of MP over a semiring $\mathbb{S} = (S, \oplus, \odot, \mathbb{0}, \mathbb{1})$ and an alphabet Σ are defined by the following grammar:

$$\varphi \ := \ s \mid (\varphi \oplus \varphi) \mid (\varphi \odot \varphi) \mid \bigoplus \mathtt{R}.\ \varphi \mid \bigodot \mathtt{R}.\ \varphi$$

where $s \in \mathbb{S}$ and $\mathtt{R} \in \mathrm{RS}_\Sigma$ is a regular selector. Similar as in [7], our formulas use constants $s \in \mathbb{S}$ and moreover constants are the only atomic formulas in MP. Our logic also includes the binary sum $\oplus$ and product $\odot$ like it is common in weighted or quantitative logics [7, 14]. Of course, the signature of these operators depends on the semiring that is chosen, for example $\max\{\varphi_1, \varphi_2\}$ or $\varphi_1 + \varphi_2$ are MP-formulas for the max-plus semiring $\mathbb{N}_{-\infty}(\max, +)$. The new quantifiers here are the formulas of the form $\bigoplus \mathtt{R}.\ \varphi$ or $\bigodot \mathtt{R}.\ \varphi$. We say that $\bigoplus \mathtt{R}.$ and $\bigodot \mathtt{R}.$ are partition quantifiers. We stress again that the signature of these quantifiers depends on the signature of the semiring. The idea here is that, over any input $w \in \Sigma^*$, $\mathtt{R}$ will select the set of maximal intervals I of w and then φ will be computed over each substring $w[i,j]$ for $(i,j) \in I$. The outputs of φ over $w[i,j]$ will be aggregated under the $\oplus$ or $\odot$ operation. It is important to remark that φ will be computed over a substructure of w and not over the whole string. This differs from the classical logic semantics where an element, set or relation is chosen and the subformulas are evaluated over the whole structure plus an assignment over the variables. Here we have taken a different direction and we consider just the substructure induced by the interval provided by the regular selector.

Formally, each MP-formula φ defines a function $[\![\varphi]\!]$ from Σ^* to $\mathbb{S}$. The semantics of MP-formulas is defined recursively over any string $w \in \Sigma^*$ as follows:

$$\begin{aligned}
[\![s]\!](w) &:= s \\
[\![\varphi_1 \oplus \varphi_2]\!](w) &:= [\![\varphi_1]\!](w) \oplus [\![\varphi_2]\!](w) \\
[\![\varphi_1 \odot \varphi_2]\!](w) &:= [\![\varphi_1]\!](w) \odot [\![\varphi_2]\!](w) \\
[\![\textstyle\bigoplus \mathtt{R}.\ \varphi]\!](w) &:= \bigoplus_{(i,j) \in \mathrm{Max}(w,\mathtt{R})} [\![\varphi]\!](w[i,j]) \\
[\![\textstyle\bigodot \mathtt{R}.\ \varphi]\!](w) &:= \bigodot_{(i,j) \in \mathrm{Max}(w,\mathtt{R})} [\![\varphi]\!](w[i,j])
\end{aligned}$$

for any MP-formulas φ, φ_1, and φ_2; and for any regular selector $\mathtt{R}$ over Σ. For the special case when $\mathrm{Max}(w, \mathtt{R}) = \varnothing$, we define $[\![\bigoplus \mathtt{R}.\ \varphi]\!](w) = \mathbb{0}$ and $[\![\bigodot \mathtt{R}.\ \varphi]\!](w) = \mathbb{1}$.

In the sequel we give some examples in order to understand the syntax and semantics of the logic.

▶ **Example 5.** Suppose that we want to compute the number of b-symbols in a string and we want to specify this function with MP-formulas over the max-plus semiring $\mathbb{N}_{-\infty}(\max,+)$. Here, we use $\max\{\cdot,\cdot\}$ and $+$ for the binary operators, and $\text{Max}\ \mathtt{R}.\ \varphi$, $\sum \mathtt{R}.\ \varphi$ for the partition quantifiers. Then the number of b-symbols in a string can be computed easily with the following formula:

$$\varphi_1 \ := \ \sum\ b.\ 1$$

To understand φ_1, we need to first understand the regular selector given by the simple expression b. Recall that this is a shorthand for $(a+b)^*\langle b\rangle(a+b)^*$. Thus, the regular selector b is choosing all the maximal intervals with just one b-symbol, that is, all substrings of the form b. Then for each b-symbol in the input the formula is outputting 1 and, by aggregating them all, it is calculating the number of b-symbols in a string.

By definition for any fixed string u, a formula of the form $\sum u.\ 1$ counts how many times the u-string appears in the input. It is interesting to compare how simple and readable is this formula in comparison to any equivalent formula in other logics (e.g. weighted logics [7]) or other formalism (e.g. weighted expressions [20]) for computing function over strings.

MP also has the ability of defining regular properties in a simple way. For example, let R be a regular expression and suppose one wants to output $\mathbb{1}$ if the input is definable by R and $\mathbb{0}$ otherwise. This is defined by the expression $\bigoplus \epsilon\langle R\rangle\epsilon.\ \mathbb{1}$. Here, the prefix and suffix of the selected interval are ϵ, thus the regular selector chooses the whole string depending if it belongs to R. If the string belongs to R the formula outputs $\mathbb{1}$; otherwise it outputs $\mathbb{0}$. Therefore, MP has a native use of regular expressions embedded in the language.

▶ **Example 6.** Suppose that one wants to compute the length of the maximum substring of b-symbols. The following formula shows how to define this function in MP logic over the semiring $\mathbb{N}_{-\infty}(\max,+)$:

$$\varphi_2 \ := \ \text{Max}\ \ b^+.\ \sum b.\ 1$$

In the previous formula, the partition quantifier $\text{Max}\ \ b^+$ is breaking the input into maximal substrings of b-symbols and passing each substring to the subformula $\sum b.\ 1$ that counts the number of b-symbols in the substring. Finally we maximize over all maximal substrings of b-symbols.

We want to highlight again how declarative is φ_2 in comparison to other logics. Here the words are partitioned into maximal substrings of b-symbols and the length of each substring is counted. In the end it is maximized over all lengths.

The next example defines a more complicated function.

▶ **Example 7.** Let $\Sigma = \{a, b, \#\}$ and suppose that we want to compute the same function as in Example 4, that is, for each subinterval between #-letters, we want the maximum between its number of a- or b-symbols, and then sum these values over all intervals. This complicated function can be easily defined by the following MP formula over the max-plus semiring $\mathbb{N}_{-\infty}(\max,+)$:

$$\varphi_3 \ := \ \sum (a+b)^+.\ \max\left\{\ \sum a.\ 1\ ,\ \sum b.\ 1\ \right\}$$

One can easily understand the function from the definition of the MP-formula φ_3. The first quantifier $\sum (a+b)^+$ is dividing the word into maximal substrings of a- and b-symbols or, in other words, substrings that are between #-symbols (or the prefix and the suffix). Then for each of these substrings the subformula $\max\left\{\ \sum a.\ 1\ ,\ \sum b.\ 1\ \right\}$ is taking the maximum between the number of a-symbols or b-symbols. In the end these values are summed over all maximal substrings of a- and b-symbols.

3.3 Design decisions behind MP

MP uses regular selectors for choosing intervals from the input and computing a subformula over the selected substrings. Here we are taking two design decisions about this new logic: (1) we decided to use regular expressions for selecting intervals and (2) we consider only the maximal intervals. In the following we give evidence of how these decisions are related with previous work.

Regular expressions have been used from the beginning for extracting intervals from strings [1, 10]. For example, regular expressions are used in practice for matching substrings from files or documents [10]. Similar to regular selectors, a RegExp-engine (like egrep) parses a regular expression R and an input document D, and extracts all words from D that match with R. RegExp-engines even use parentheses "$(\cdot)$" for declaring that the subword that matches the subexpressions between parentheses must be output. Furthermore, in RegExp-engines the parentheses semantics is greedy, namely, they select the larger subword that matches the subexpression inside parentheses. This semantics is similar to the maximal semantics of regular selectors with the exception that the greedy-semantics is even more restrictive since the selected interval depends on how the input is parsed from left-to-right [10]. Despite this fact, it is interesting that even a more restricted flavor of the maximal semantics is already presented in practice which supports the decision of including it for MP.

Recently, regular expressions for substring selection have been considered in the context of information extraction [9, 12]. In [9], the authors propose a regular expression language enhanced with variables, called regex, to extract relations of substrings from an unstructured document. Regular selectors can be seen as a restrictive subfragment of regex, where only one variable is used. We note that we could have used regex language or any other formalism with the maximal semantics for selecting intervals from a string. However, we believe that regular selectors are very simple, flexible and concise, and they include the best features of previous works without loosing expressibility [9].

Finally, we could have also chosen MSO logic with two free variables for selecting intervals instead of regular expressions (i.e. with respect to the normal semantics), namely, for any MSO-formula $\varphi(x,y)$ to extract the set $\mathrm{Sel}(w,\varphi(x,y))$ of all intervals (i,j) over a string w such that: $w \vDash \varphi(i,j)$. Of course, both formalism for selecting intervals are equivalent. Namely, it is easy to show that for every MSO-formula $\varphi(x,y)$ there exists a finite set of regular selectors $\mathtt{R}_1,\ldots,\mathtt{R}_n$ such that $\bigcup_{i=1}^{n}\mathrm{Sel}(w,\mathtt{R}_i) = \mathrm{Sel}(w,\varphi(x,y))$ and vice versa. Notice that with this definition of selecting intervals by MSO formulas we can assume that formulas additionally satisfy $x \leq y$.

Regarding the maximal semantics of regular selectors, it is important to note that a similar semantics was studied before. In [5] the authors define a subset of MSO formulas with two free variables called rigid MSO-formulas. Formally, an MSO-formula $\varphi(x,y)$ over strings is called rigid if for all strings $w \in \Sigma^*$ and all positions $i \in \mathrm{dom}(w)$ there is at most one position $j \in \mathrm{dom}(w)$ such that $w \vDash \varphi(i,j)$, and at most one $j' \in \mathrm{dom}(w)$ such that $w \vDash \varphi(j',i)$; in other words, $\varphi(x,y)$ defines two partial injective functions on $\mathrm{dom}(w)$. One can easily check that intervals defined by a regular selector with the maximal semantics are also definable by a rigid MSO-formula. Indeed, for any regular selector $\mathtt{R}$ suppose that $\varphi_{\mathtt{R}}(x,y)$ is an equivalent MSO formula that defines the same set of intervals (i.e. with the normal semantics). Then $\mathrm{Max}(w,\mathtt{R}) = \mathrm{Sel}(w,\varphi^*_{\mathtt{R}}(x,y))$, where:

$$\varphi^*_{\mathtt{R}}(x,y) \;:=\; \varphi_{\mathtt{R}}(x,y) \;\wedge\; \forall x'.\forall y'.\; (\varphi_{\mathtt{R}}(x',y') \;\wedge\; x' \leq x \;\wedge\; y \leq y') \rightarrow (x' = x \;\wedge\; y' = y)$$

The formula $\varphi^*_{\mathtt{R}}(x,y)$ is restricting the intervals that satisfy $\varphi_{\mathtt{R}}(x,y)$ to be maximal. In particular, one can easily check that $\varphi^*_{\mathtt{R}}(x,y)$ is indeed a rigid formula. This implies that

the maximal semantics can be expressed by rigid formulas. The next proposition shows that rigid formulas can also be defined by sets of regular selectors with the maximal semantics.

▶ **Proposition 8.** *For every regular selector* $\mathtt{R}$ *there exists a rigid formula* $\varphi_{\mathtt{R}}(x,y)$ *such that* $\mathrm{Max}(w,\mathtt{R}) = \mathrm{Sel}(w,\varphi_{\mathtt{R}}(x,y))$ *for every* $w \in \Sigma^*$. *Furthermore, for every rigid formula* $\varphi(x,y)$ *there exists a set of regular selectors* $\mathtt{R}_1,\ldots,\mathtt{R}_n$ *such that* $\mathrm{Sel}(w,\varphi(x,y)) = \bigcup_{i=1}^{n} \mathrm{Max}(w,\mathtt{R}_i)$ *for every* $w \in \Sigma^*$.

4 Automata-based characterization of MP

In [17] (see Corollary 1) it was shown that the class of functions defined by copyless CRA is not closed under reverse, that is, the run of copyless CRA is asymmetric with respect to the input. Intuitively, this fact is contrary to the spirit of a logical characterization for a computational model: a logic should express properties over the whole string and its expressiveness should not depend on the orientation of the input. This implies that a characterization of copyless CRA in terms of a logic is far to be possible. To solve this, we introduce the subclass of *bounded alternation copyless* CRA (in short BAC) which is a restricted variant of copyless CRA. We show that BAC have good closure properties and, moreover, this is the right model to capture the expressiveness of maximal partition logic.

The *alternation* of an expression $e \in \mathrm{Expr}(\mathcal{X})$ is defined as the maximum number of switches between $\oplus$ and $\odot$ operations over all branches of the parse-tree of e. Formally, let $\otimes \in \{\oplus,\odot\}$ and $\bar{\otimes}$ be the dual operation of $\otimes$ in $\mathbb{S}$. We define the set of expressions $\mathrm{Expr}_0^{\otimes}(\mathcal{X})$ with 0-alternation by $\mathrm{Expr}_0^{\otimes} = \mathcal{X} \cup \mathbb{S}$. For any $N \geq 1$, we define the set of expressions $\mathrm{Expr}_N^{\otimes}(\mathcal{X})$ as the $\otimes$-closure of $\mathrm{Expr}_{N-1}^{\otimes}(\mathcal{X})$, namely, $\mathrm{Expr}_N^{\otimes}(\mathcal{X})$ is the minimal set of expressions that contains $\mathrm{Expr}_{N-1}^{\otimes}(\mathcal{X})$ and satisfies $e_1 \otimes e_2 \in \mathrm{Expr}_N^{\otimes}(\mathcal{X})$ for all $e_1, e_2 \in \mathrm{Expr}_N^{\otimes}(\mathcal{X})$. We denote by $\mathrm{Expr}_N(\mathcal{X}) = \mathrm{Expr}_N^{\oplus}(\mathcal{X}) \cup \mathrm{Expr}_N^{\odot}(\mathcal{X})$ the set of all expressions with alternation bounded by N.

We say that a copyless CRA $\mathcal{A}$ has *bounded alternation* if there exists $N \in \mathbb{N}$ such that for every $w \in \Sigma^*$ it holds that $|\mathcal{A}|(w) \in \mathrm{Expr}_N(\mathcal{X})$, that is, the number of alternations of all ground expressions output by $\mathcal{A}$ is uniformly bounded by a constant. A copyless CRA $\mathcal{A}$ is called a bounded alternation copyless CRA (in short BAC) if $\mathcal{A}$ has bounded alternation. All the examples of copyless CRA presented in this paper have bounded alternation. For example, functions in Examples 1 and 2 are part of the BAC-class.

Bounding the alternation of expressions or formulas is a standard assumption in logic [16] and here we used it to syntactically restrict the expression constructed by a copyless CRA. One can easily check that this syntactical property can be verified in NLogSpace in the size of the copyless CRA. Indeed, a copyless CRA has unbounded alternation iff there exists a loop that alternates between $\odot$ and $\oplus$ in its transition graph. Of course, the existence of such loops can be determined by standard reachability tests in NLogSpace [19].

The fact that we can express the BAC automata in Examples 1 and 2 by MP-formulas in Examples 6 and 7 , respectively, is not a coincidence. In the following theorem, we present the main result of the paper.

▶ **Theorem 9.** *Maximal partition logic and bounded alternation copyless CRA are equally expressive, that is:*

- *for every* MP*-formula* φ *there exists a BAC* $\mathcal{A}_\varphi$ *such that* $[\![\varphi]\!] = [\![\mathcal{A}_\varphi]\!]$;
- *for every BAC* $\mathcal{A}$ *there exists a* MP*-formula* $\varphi_{\mathcal{A}}$ *such that* $[\![\mathcal{A}]\!] = [\![\varphi_{\mathcal{A}}]\!]$.

Proof. We present here sketch proofs of the two directions. Due to the space limit, the full proofs are moved to the appendix (available online).

From logic to automata. Let φ be a MP-formula. We sketch the definition of a BAC $\mathcal{A}$ that specifies the same function as φ. The proof is by induction over the size of φ. The interesting case is when $\varphi = \otimes R\langle S\rangle T.\ \psi$ where $R\langle S\rangle T$ is a regular selector and ψ is an MP-formula for which there exists a BAC $\mathcal{B}$ such that $[\![\psi]\!] = [\![\mathcal{B}]\!]$. The main idea behind the definition of $\mathcal{A}$ is to keep many copies of the automaton $\mathcal{B}$ and each copy is responsible for evaluating the formula ψ on intervals defined by $R\langle S\rangle T$. For $\otimes$-aggregating the outputs of the $\mathcal{B}$-copies, $\mathcal{A}$ uses one additional register x^* that, each time an interval is closed, the output of the $\mathcal{B}$-copy is $\otimes$-operated with x^* and then stored in x^*.

Let $\mathcal{A}_R$, $\mathcal{A}_S$, and $\mathcal{A}_T$ be the finite automata recognizing the regular languages R, S, and T, respectively. The first issue we have to deal with is that the number of $\mathcal{B}$-copies cannot depend on the input string w. We prove that, for every k-position in w, the number of maximal intervals defined by $R\langle S\rangle T$ and containing k is uniformly bounded. Moreover this bound is universal for all strings, i.e., it depends only on the size of $\mathcal{A}_S$. To see this, suppose that I is the set of maximal intervals defined by $R\langle S\rangle T$ and containing k. Furthermore, suppose that the size of I is bigger than the number of states in $\mathcal{A}_S$. If we assign to every interval in I the state of $\mathcal{A}_S$ in position k, then there are two intervals i_1 and i_2 with the same state assigned. It is easy to see that we can merge these two intervals into one interval that is selected by $R\langle S\rangle T$ but is bigger than i_1 and i_2. This is clearly a contradiction with the fact that both i_1 and i_2 are maximal.

The second issue is to recognize the maximal intervals selected by $R\langle S\rangle T$ while $\mathcal{A}$ is reading the input. The main observation here is that one can rewrite $R\langle S\rangle T$ into a new regular selector that does not needs maximal semantics, i.e., there exists a regular selector $R'\langle S'\rangle T'$ that defines with the normal-semantics all maximal intervals selected by $R\langle S\rangle T$. Thus, we can assume that $R\langle S\rangle T$ is already in this form and we focus on all selected intervals.

Now that $\mathcal{A}$ does not have to deal with checking whether an interval is maximal or not, it has to decide whether an interval will be selected by $R\langle S\rangle T$. Of course, $\mathcal{A}$ can keep track of runs of $\mathcal{A}_R$, $\mathcal{A}_S$, and $\mathcal{A}_T$ over w to find new potential intervals selected by $R\langle S\rangle T$. The problem is that, in the end, the intervals can turn out to be spurious (e.g. the remaining suffix does not belong to the language defined by T) and we cannot afford to keep all potential intervals since the number of $\mathcal{B}$-copies is bounded. To deal with this issue we use Theorem 2 in [17] which shows that BAC are closed under regular-lookahead, that is, the model can be extended with regular look-ahead and this does not add more expressibility to the model. This extension allows BAC to make decisions based on whether the remaining suffix of the input word belongs to a regular language or not. By using this extension, $\mathcal{A}$ can determine in advance whether an interval is going to be selected by $R\langle S\rangle T$ and solve the problem with the spurious intervals.

The final automaton $\mathcal{A}$ works as follows. Whenever $\mathcal{A}$ finds a new interval selected by $R\langle S\rangle T$, it starts evaluating a $\mathcal{B}$-copy over this interval. With regular look-ahead it also checks if an interval is closing. If that is the case, then the output of the $\mathcal{B}$-copy in charge of this interval is aggregated with the additional register x^* and the registers in this $\mathcal{B}$-copy are reset to the values defined by the initial function of $\mathcal{B}$. Finally, the output function of $\mathcal{A}$ is defined by aggregating x^* with all intervals closed in the last step of $\mathcal{A}$.

From automata to logic. Let $\mathcal{A} = (Q, \Sigma, \mathcal{X}, \delta, q_0, \nu_0, \mu)$ be a bounded alternation copyless CRA. We sketch the definition of the formula $\varphi_{\mathcal{A}}$ that defines the same function as $\mathcal{A}$. The proof is by induction over the alternation bound N of $\mathcal{A}$.

The first step is to understand the ground expressions defined by $\mathcal{A}$. Let g be the ground expression defined by the run of $\mathcal{A}$ on a string w. By applying the associativity and

commutativity of $\mathbb{S}$, one can show that g can be rewritten into an expression g^* of the form $\bigotimes_{c \in C} c \otimes \bigotimes_{e \in E} e$ for some operation $\otimes \in \{\oplus, \odot\}$, where $C \subseteq S$ is a multiset of constants and E is a multiset of expressions whose alternation is strictly lower than N. Interestingly, one can define MP-formulas $\varphi_C^{\otimes}$ and $\varphi_E^{\otimes}$ each taking care of $\bigotimes_{c \in C} c$ and $\bigotimes_{e \in E} e$, respectively. To define $\varphi_C^{\otimes}$, we use a set of regular selectors that chooses all 1-letter intervals where each constant in C was generated by a transition of $\mathcal{A}$. Here we define the selectors in such a way that in each position we are able to retrieve the state and substitution used in the run of $\mathcal{A}$. The formula $\varphi_C^{\otimes}$ is then defined by aggregating the right constants (i.e. the ones in C) used by substitutions of the run of $\mathcal{A}$ over w.

The formula $\varphi_E^{\otimes}$ requires more effort. For every expression $e \in E$ we define a BAC $\mathcal{A}_e$, a modified variant of $\mathcal{A}$, such that $\mathcal{A}_e$ outputs e on a substring $w[i_e, j_e]$. We modify only q_0, ν_0, and μ, and the other components $\mathcal{X}$, Q and δ remain the same. Thus, the number of new automata does not depend on the size of E but only on $\mathcal{A}$. Given that the expressions in E have alternation strictly less than N, then by induction we can find a formula $\varphi_{\mathcal{A}_e}$ for every automaton $\mathcal{A}_e$. The main difficulty in the proof is to define regular selectors that find the intervals (i_e, j_e), where $\mathcal{A}_e$ or, more concretely $\varphi_{\mathcal{A}_e}$, must be applied. Indeed, it is easy to define a set of expressions that find these intervals but the problem is the maximal semantic, in particular, the set of intervals $\{(i_e, j_e) \mid e \in E\}$ does not have to be a set of maximal intervals. To solve this problem we define the intervals by rigid formulas instead of using the maximal semantics. By Proposition 8, one can turn a rigid formula into a sum of selectors that define the same set of intervals on every string.

Summing up, having the formulas $\varphi_C^{\otimes}$ and $\varphi_E^{\otimes}$ defined, it is easy to define the final formula $\varphi_{\mathcal{A}}$. Notice that for the base cases of the induction (i.e. when $N = 0, 1$) we do not need the formula $\varphi_E^{\otimes}$ and, therefore, $\varphi_C^{\otimes}$ includes the base case. Of course, there are some exceptional cases not discussed in this proof-sketch because of space restrictions. The full proof includes all these cases. ◀

Theorem 9 gives a logic-based characterization of bounded alternation copyless CRA. This is useful to show new results in the automata model that are implications from the logic counterpart. For example, one can easily show that MP is invariant under the orientation of a word.

▶ **Proposition 10.** *For every formula φ in* MP *there exists a formula φ^r such that for all words $[\![\varphi]\!](w) = [\![\varphi^r]\!](w^r)$, where w^r is the reverse word of w.*

Interestingly, Proposition 10 and Theorem 9 implies that the BAC-class is closed under reverse. Note that this result is unexpected if we try to prove it directly from the automata model.

▶ **Corollary 11.** *For every BAC $\mathcal{A}$ there exists a BAC $\mathcal{A}^r$ that computes the reverse function, that is, $[\![\mathcal{A}]\!](w) = [\![\mathcal{A}^r]\!](w^r)$ for every $w \in \Sigma^*$.*

The logic-based characterization of BAC and its good closure properties suggest that these automata are a robust class in the world of weighted automata. In the next section, we compare its expressibility with respect to weighted MSO and weighted automata.

5 Weighted MSO vs MP

In this section we compare MP with Weighted MSO, a quantitative logic that was proposed as the logic counterpart of weighted automata. Recall that formulas of Weighted MSO [7]

(WMSO) over a semiring $\mathbb{S} = (S, \oplus, \odot, \mathbb{0}, \mathbb{1})$ and an alphabet Σ are defined by the following grammar (note that we use the modern syntax from [4, 14]):

$$\theta \coloneqq \varphi \mid s \mid (\theta \oplus \theta) \mid (\theta \odot \theta) \mid \bigoplus x.\ \theta(x) \mid \bigodot x.\ \theta(x) \mid \bigoplus X.\ \theta(X)$$

where φ is an MSO-formula over Σ, $s \in \mathbb{S}$, x is a first-order variable, and X is a set of variables. The syntax of WMSO is given by boolean formulas (for the MSO fragment) and quantitative formulas (for the rest of the syntax). Let $w = w_1 \dots w_n$ be a string over Σ and σ a $(\bar{x}, w)$-assignment. The semantics $[\![\varphi]\!](w, \sigma)$ of a boolean formula φ over w and σ is equal to $\mathbb{1}$ if $w \vDash \varphi(\sigma)$ and $\mathbb{0}$ otherwise. The semantics of a quantitative formula θ over w and σ is defined as follows.

$$\begin{aligned}
[\![s]\!](w,\sigma) &\coloneqq s \\
[\![(\theta_1 \otimes \theta_2)]\!](w,\sigma) &\coloneqq [\![\theta_1]\!](w,\sigma) \otimes [\![\theta_2]\!](w,\sigma) && \text{for } \otimes \in \{\oplus, \odot\} \\
[\![\bigotimes x.\ \theta(x)]\!](w,\sigma) &\coloneqq \bigotimes_{i=1}^{n} [\![\theta(x)]\!](w, \sigma[x \to i]) && \text{for } \otimes \in \{\oplus, \odot\} \\
[\![\bigoplus X.\ \theta(X)]\!](w,\sigma) &\coloneqq \bigoplus_{I \subseteq [1,n]} [\![\theta(X)]\!](w, \sigma[X \to I])
\end{aligned}$$

▶ **Example 12.** One can compare WMSO with MP by defining WMSO formulas for the functions in Examples 5 and 6. We start with the WMSO-formula for counting the number of b-symbols in a string:

$$\sum x. \max\{P_b(x) + 1, 0\} \tag{1}$$

To understand formula (1), recall that in the semiring $\mathbb{N}_{-\infty}(\max, +)$ the operations and constants are defined as follows: $\mathbb{0} = -\infty$, $\mathbb{1} = 0$, $\oplus = \max$ and $\odot = +$. For any position i and assignment $x \to i$, if i is labeled with b then $P_b(x)$ evaluates to 0; otherwise $P_b(x)$ evaluates to $-\infty$. Now it is easy to understand formula (1): we are summing 1 over all positions with a b-symbol and 0 over all other positions.

To define the length of the maximum substring of b-symbols, as in Example 6, one can write the following WMSO-formula:

$$\mathrm{Max}\, x. \sum y. \max\left\{(x \le y \wedge \forall z.(x \le z \wedge z \le y) \to P_b(z)) + 1, 0\right\} \tag{2}$$

The formula (2) selects all pairs (x, y). The boolean subformula is satisfied if (x, y) is an interval of b's; then such a pair contributes 1, otherwise it contributes 0. For a fixed x the formulas sums over all y that vary through all elements of the interval (x, y). Since we take maximum over all variables x, we get the desired formula.

WMSO was proposed by Droste and Gastin as the logic counterpart of weighted automata but it turns out to be more expressive. In [7] it is shown that by restricting the nesting and alternation of semiring quantifiers $\bigoplus x$, $\bigodot x$, and $\bigoplus X$ one can capture exactly the expressiveness of weighted automata. For more details we refer the reader to the paper [7]. We shall use their notation to define different fragments of WMSO. The fragment of WMSO equally expressive to weighted automata is denoted $\mathrm{WMSO}[\bigoplus_X \bigodot_x^1]$. Furthermore, in [14] it was shown that two natural fragments of weighted automata, namely, finitely ambiguous weighted automata and polynomial ambigous weighted automata are equally expressive to the fragments denoted respectively by $\mathrm{WMSO}[\bigodot_x^1]$ and $\mathrm{WMSO}[\bigoplus_x \bigodot_x^1]$. By results in [13, 14] this shows that in terms of expressiveness, these fragments are strictly contained in each other:

$$\mathrm{WMSO}[\bigodot_x^1] \subsetneq \mathrm{WMSO}[\bigoplus_x \bigodot_x^1] \subsetneq \mathrm{WMSO}[\bigoplus_X \bigodot_x^1]$$

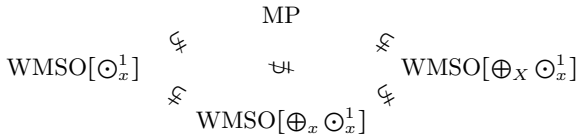
We compare the expressiveness of MP with WMSO by exploiting the relation with copyless CRA (Theorem 9). The first question is whether MP is more expressive than $\mathrm{WMSO}[\bigoplus_X \odot_x^1]$. At a first sight, one could believe that this is possible, since the syntax of MP is symmetric with respect to both semiring operations, that is, there is no syntactical restriction on $\oplus$- and $\otimes$-quantifiers. Interestingly, in terms of expressiveness, MP is contained in $\mathrm{WMSO}[\bigoplus_X \odot_x^1]$. We prove this by showing that functions definable by copyless CRA are also definable by weighted automata. This result combined with Theorem 9 proves the following proposition.

▶ **Proposition 13.** *For every formula in* MP *there exists a formula in* $\mathrm{WMSO}[\bigoplus_X \odot_x^1]$ *defining the same function.*

The previous upper-bound opens the question of what is a good lower-bound for the expressiveness of MP. An answer to this question is given in the next result which shows that MP contains the fragment $\mathrm{WMSO}[\odot_x^1]$. We prove this result (see the appendix) by showing that every function definable by a finitely ambiguous weighted automaton is definable by a bounded alternation copyless CRA. This combined with the results in [14] and Theorem 9 proves the next proposition.

▶ **Proposition 14.** *For every formula in* $\mathrm{WMSO}[\odot_x^1]$ *there exists a formula in* MP *defining the same function.*

The examples presented in Section 3 tell us a bit more about the expressiveness of MP. For example, it was shown in [13] that the function from Example 4 is not definable by any finitely ambiguous weighted automata. This proves that $\mathrm{WMSO}[\odot_x^1]$ is strictly contained in MP. On the other hand, in [17] it is shown that there exists a function that is definable by polynomial ambiguous weighted automata but it is not definable by any copyless CRA. This shows that MP is strictly contained in $\mathrm{WMSO}[\bigoplus_X \odot_x^1]$ and, moreover, is does not contain $\mathrm{WMSO}[\bigoplus_x \odot_x^1]$. Summing up, we get the following diagram representing the expressiveness of MP in terms of WMSO.



We conjecture that MP is not contained in $\mathrm{WMSO}[\bigoplus_x \odot_x^1]$, such a result would complete the diagram. We guess that this can be shown by proving that the function from Example 2 is not definable by any polynomial ambiguous weighted automata.

6 Conclusions and future work

In this paper we proposed and investigated maximal partition logic. Our main result shows that MP is a logic characterization of BAC, a natural restriction of copyless CRA. MP has no syntactical restrictions and, in contrast to Weighted MSO, there is no division between the boolean and the quantitative parts of the logic. A mild restriction is put in the semantics of the logic since we allow only maximal intervals. Thanks to this semantic our formulas are usually more readable and easy to write (see Example 3).

For future work we would like to extend MP and copyless CRA beyond semirings. It seems that in our proofs we need the commutativity, associativity and the neutral element of each operator separately, but we do not use the distributivity. For this reason we think that we could extend the semiring with additional operators and the results proved in this work

will still hold. The comparison of MP with WMSO shows that this logic is in the edge of decidability. It lays between finite ambiguous weighted automata, a class of functions with good decidability properties, and weighted automata for which most interesting problems are undecidable. For this reason, we believe that for future work it is important to understand the decidability properties of MP and copyless CRA.

Acknowledgments. We thank the anonymous referees for their helpful comments. In particular for the comment simplifying the proof of Theorem 9. The first author was supported by Poland's National Science Center grant 2013/09/N/ST6/01170. The last author was supported by CONICYT + PAI / Concurso Nacional Apoyo al Retorno de Investigadores/as desde el extranjero – Convocatoria 2013 + 821320001 and by the Millenium Nucleus Center for Semantic Web Research under grant NC120004.

References

1 V Alfred. Algorithms for finding patterns in strings. *Handbook of Theoretical Computer Science: Algorithms and complexity*, 1:255, 1990.

2 Shaull Almagor, Udi Boker, and Orna Kupferman. What's decidable about weighted automata? In *ATVA*, pages 482–491, 2011.

3 Rajeev Alur, Loris D'Antoni, Jyotirmoy Deshmukh, Mukund Raghothaman, and Yifei Yuan. Regular functions and cost register automata. In *Proceedings of the 2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 13–22. IEEE Computer Society, 2013.

4 Benedikt Bollig, Paul Gastin, Benjamin Monmege, and Marc Zeitoun. Logical characterization of weighted pebble walking automata. In *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, page 19. ACM, 2014.

5 Thomas Colcombet, Clemens Ley, and Gabriele Puppis. On the use of guards for logics with data. In *Mathematical Foundations of Computer Science 2011*, pages 243–255. Springer, 2011.

6 Karel Culik II and Jarkko Kari. Image compression using weighted finite automata. In *Mathematical Foundations of Computer Science 1993*, pages 392–402. Springer, 1993.

7 Manfred Droste and Paul Gastin. Weighted automata and weighted logics. *Theor. Comput. Sci.*, 380(1-2):69–86, 2007.

8 Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of Weighted Automata*. Springer, 1st edition, 2009.

9 Ronald Fagin, Benny Kimelfeld, Frederick Reiss, and Stijn Vansummeren. Spanners: a formal framework for information extraction. In *Proceedings of the 32nd symposium on Principles of database systems*, pages 37–48. ACM, 2013.

10 Jeffrey Friedl. *Mastering regular expressions*. " O'Reilly Media, Inc.", 2006.

11 John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

12 Benny Kimelfeld. Database principles in information extraction. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 156–163. ACM, 2014.

13 Ines Klimann, Sylvain Lombardy, Jean Mairesse, and Christophe Prieur. Deciding unambiguity and sequentiality from a finitely ambiguous max-plus automaton. *Theor. Comput. Sci.*, 327(3):349–373, 2004.

14 Stephan Kreutzer and Cristian Riveros. Quantitative monadic second-order logic. In *Proceedings of the 2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 113–122. IEEE Computer Society, 2013.

15 Daniel Krob. The equality problem for rational series with multiplicities in the tropical semiring is undecidable. In *ICALP*, pages 101–112, 1992.

16 Leonid Libkin. *Elements of finite model theory*. Springer Science & Business Media, 2004.

17 Filip Mazowiecki and Cristian Riveros. On the expressibility of copyless cost register automata. *CoRR*, abs/1504.01709, 2015.

18 Mehryar Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311, 1997.

19 Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1993.

20 Jacques Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, 2009.

21 M. P. Schützenberger. On the definition of a family of automata. *Information and Control*, 4:245–270, 1961.

Aperiodic Two-way Transducers and FO-Transductions

Olivier Carton[1] and Luc Dartois[2,3]

1 **LIAFA, Université Paris Diderot, France**
`Olivier.Carton@liafa.univ-paris-diderot.fr`
2 **LIF, UMR7279 Aix-Marseille Université & CNRS, France**
`luc.dartois@lif.univ-mrs.fr`
3 **Centrale Marseille, France**

Abstract

Deterministic two-way transducers on finite words have been shown by Engelfriet and Hoogeboom to have the same expressive power as **MSO**-transductions. We introduce a notion of aperiodicity for these transducers and we show that aperiodic transducers correspond exactly to **FO**-transductions. This lifts to transducers the classical equivalence for languages between **FO**-definability, recognition by aperiodic monoids and acceptance by counter-free automata.

1998 ACM Subject Classification F.4.3 Formal Languages

Keywords and phrases Transducer, first-order, two-way, transition monoid, aperiodic

Digital Object Identifier 10.4230/LIPIcs.CSL.2015.160

1 Introduction

The regularity of a language of finite words is a central notion in theoretical computer science. Combining several seminal results, it is equivalent whether a language is
(a) accepted by a (non-)deterministic one-way or two-way automaton [22] and [27],
(b) described by a regular expression [17],
(c) defined in (Existential) Monadic Second Order (**MSO**) logic [8],
(d) the preimage by a morphism into a finite monoid [20].
Since then, the characterization of fragments of **MSO** has been a very successful story. Using this equivalence between different formalisms, several fragments of **MSO** have been characterized by algebraic means and shown to be decidable. Combining results of Schützenberger [25] and of McNaughton and Papert [19] yields, for instance, that a language of finite words is First Order (**FO**) definable if and only if all the groups contained in its syntactic monoid are trivial (aperiodic). From the results of Schützenberger [26] and others [28], it is also known that a language is First Order definable with two variables ($\mathbf{FO}^2$) if and only if its syntactic monoid belongs to the class **DA** which is easily decidable.

Automata can be equipped with output to make them compute functions and relations. They are then called transducers. Note then that all variants are no longer equivalent as they are as acceptors. Deterministic transducers compute a subclass of rational functions called sequential functions [9]. Two-way transducers are also more powerful than one-way transducers (see Example 1). The study of transducers has many applications. Transducers are used to model coding schemes (compression schemes, convolutional coding schemes, coding schemes for constrained channels, for instance). They are also widely used in computer arithmetic [15], natural language processing [24] and programs analysis [11].

24th EACSL Annual Conference on Computer Science Logic (CSL 2015).
Editor: Stephan Kreutzer; pp. 160–174

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The equivalence between automata and **MSO** has been first lifted to transducers and the functions they realize by Engelfriet and Hoogeboom [13]. They show that a function from words to words can be realized by a deterministic two-way transducer if and only it is a **MSO**-transduction. First, this result deals surprisingly with two-way transducers rather than one-way transducers which are much simpler. Second, the **MSO**-definability used for automata is replaced by **MSO** graphs transductions defined by Courcelle [12]. A **MSO**-transduction is a function where the output graph is defined as a **MSO**-interpretation into a fixed number of copies of the input graph. In the result of Engelfriet and Hoogeboom, words are seen as linear graphs whose vertices carry the symbols.

1.1 Contribution

In this paper, we combine the approach of Engelfriet and Hoogeboom with the one of Schützenberger, McNaughton and Papert. We introduce a notion of aperiodicity for two-way transducers and we show that it corresponds to **FO**-transductions. By **FO**-transduction, we mean **MSO**-transduction where the interpretation is done through **FO**-formulas. The definition of aperiodicity is achieved by associating a transition monoid with each two-way transducer. The construction of this algebraic object is already implicit in the literature [27, 21, 6]. In order to obtain our result, we have considered a different logical signature for transductions from the one used in [13]. In [13], the signature contains the symbol predicates to check symbols carried by vertices and the edge predicate of the graph. Since words are viewed as linear graphs, this is the same as the signature with the successor relation on words. In our result, the signature contains the symbol predicates and the order (of the linear graph). This is equivalent for **MSO**-transductions since the order can easily be defined with the successor by a **MSO**-formula. This is however not equivalent any more for **FO**-transductions that we consider. With this signature, the definition of **FO**-transduction requires that the order on the output word can be defined by a **FO**-formula. The change in the signature is necessary to obtain the result.

1.2 Related work

The aperiodic rational functions, that is, functions realized by a one-way transducer with an aperiodic transition monoid have already been characterized in [23]. This characterization is not based on logic but rather on the inverse images of aperiodic languages.

The notion of aperiodic two-way transducer was already defined and studied in [18], although their model defined length-preserving functions and the transducers had both their reading and writing heads moving two-way. The assumption that the function is length preserving makes the relation between the input and the output easier to handle.

Recently, Bojanczyk, in [7], also characterized first-order definable transducers for machines using a finer but more demanding semantic, the so-called origin semantic.

In [3], Alur and Černý defined the streaming string transducers, a one-way deterministic model equivalent to deterministic two-way transducers and **MSO** transductions. More recently, Filiot, Krishna and Trivedi proposed in [14] a definition of transition monoid for this model. They also proved that aperiodic and 1-bounded streaming string transducers have the same expressive power as **FO** transductions, which is one of the models considered by our main result.

1.3 Structure

The paper is organized as follows. Definitions of two-way transducers and FO-transductions are provided in Section 2. The construction of the transition monoid associated with a

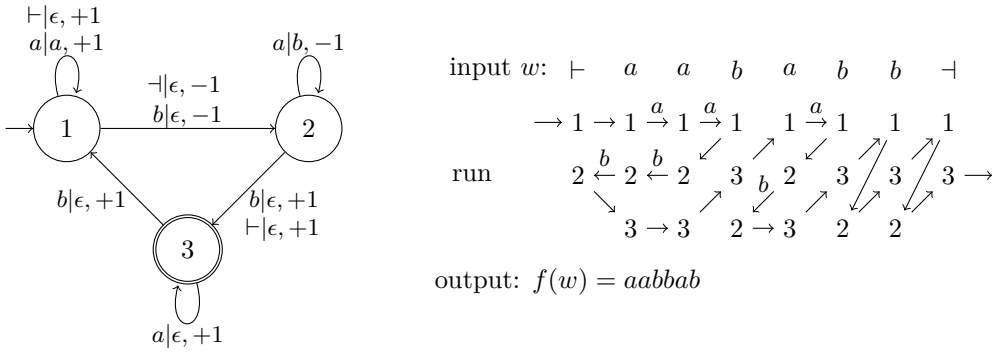


Figure 1 A transducer and its run over $w = aababb$.

transducer is given there. The main result is stated in Section 3. Section 4 focuses on one aspect of the stability by composition of functions realized by aperiodic two-way transducers. It is one of the main ingredients used in the proof of the main result. The proof itself is sketched in Sections 5 and 6.

2 Definitions

In this section, we present the different models that will be used throughout the article.

2.1 Two-way transducers

A transducer is an automaton equipped with outputs. While an input word is processed along a run by the transducer, each used transition outputs some word. All these output words are concatenated to form the output of the run. The automaton might be one-way or two-way but we mainly consider two-way transducers in this paper. When the transducer is non-deterministic, there might be several runs and therefore several output words for a single input word. All two-way transducers considered in this paper are deterministic. For each input word, there is then at most one valid run and one output word. The partial function which maps each input word to the corresponding output word is said to be *realized* by the transducer. The automaton obtained by forgetting the outputs is called the *input automaton* of the transducer.

A two-way transducer is a very restricted variant of a Turing machine with an input and an output tape. First, the input tape is read-only. Second, the output tape is write-only and the head on this tape only moves forwards. Written symbols on this tape cannot be over-written later by other symbols.

▶ **Example 1.** Let A be the alphabet $\{a, b\}$. Let us consider, as a running example, the function $f : A^* \to A^*$ which maps each word $w = a^{k_0}ba^{k_1}\cdots ba^{k_n}$ to the word $f(w) = a^{k_0}b^{k_0}a^{k_1}b^{k_1}\cdots a^{k_n}b^{k_n}$ obtained by adding after each block of consecutive a a block of consecutive b of the same length. Since each word w over A can be uniquely written $w = a^{k_0}ba^{k_1}\cdots ba^{k_n}$ with some k_i being possibly equal to zero, the function f is well defined. The word $w = aababb = a^2ba^1ba^0ba^0$ is mapped to $f(w) = a^2b^2a^1b^1a^0b^0a^0b^0 = aabbab$.

This function is realized by the transducer depicted in Figure 1. This transducer proceeds as follows to compute $f(w)$ from the input word w. While being in state 1 and moving forwards, it copies a block of consecutive a to the output. While in state 2 and moving backwards, the corresponding block of b is written to the output. While being in state 3, the

transducer moves forwards writing nothing until it reaches the next block of consecutive a. Note that this function cannot be realized by a one-way transducer.

Formally, a two-way transducer is defined as follows:

▶ **Definition 2** (Two-way transducer)**.** A (*deterministic*) *two-way transducer* $\mathcal{A}$ is a tuple $\mathcal{A} = (Q, A, B, \delta, \gamma, q_0, F))$ defined as follows:

- Q is a finite *state set*.
- A and B are the *input* and *output alphabet*.
- $\delta : Q \times (A \uplus \{\vdash, \dashv\}) \to Q \times \{-1, 0, +1\}$ is the *transition function*. Contrary to the one-way machines, the transition function also outputs an integer, corresponding to the move of the reading head. The alphabet is enriched with two new symbols $\vdash$ and $\dashv$, which are endmarkers that are added respectively at the beginning and the end of the input word, such that for all $q \in Q$, we have $\delta(q, \vdash) \in Q \times \{0, +1\}$ and $\delta(q, \dashv) \in Q \times \{-1, 0\}$.
- $\gamma : Q \times (A \uplus \{\vdash, \dashv\}) \to B^*$ is the *production function*.
- $q_0 \in Q$ is the *initial state*.
- $F \subseteq Q$ is the set of final states.

The transducer $\mathcal{A}$ processes finite words over A. If at state p the symbol a is processed and $\delta(p, a) = (q, d)$, then $\mathcal{A}$ moves to state q, moves the reading head to the left or right depending on d, and outputs $\gamma(p, a)$.

Let $w = a_1 \cdots a_n$ be a fixed finite word over A and $a_0 = \vdash$ and $a_{n+1} = \dashv$. Whenever $\delta(p, a_m) = (q, d)$ and $\gamma(p, a_m) = v$, we write $(p, m) \xrightarrow{|v} (q, n)$ where $n = m + d$. We do not write the input over the arrow because it is always the symbol below the reading head, namely, a_m. In this notation, the pairs represent the current configuration of a machine with the current state and the current position of the input head. A *run* of the transducer over w is a finite sequence of consecutive transitions

$$(p_0, m_0) \xrightarrow{|v_1} (p_1, m_1) \cdots (p_{n-1}, m_{n-1}) \xrightarrow{|v_n} (p_n, m_n)$$

and we write $(p_0, m_0) \xrightarrow{|v} (p_n, m_n)$ where $v = v_1 v_2 \cdots v_n$. We also refer to finite runs over words w when all positions m_i in the run but the last are between 1 and $|w|$. The last position m_n is allowed to be between 0 and $|w| + 1$. It is 0 if the run leaves w on the left end and it is $|w| + 1$ if it leaves $|w|$ on the right end.

A run $(p_0, m_0) \xrightarrow{|v} (p_n, m_n)$ over a marked word $\vdash u \dashv$ is *accepting* if it starts at the first position in the initial state and ends on the right endmarker $\dashv$ in a final state. Then v is the *image* of u by $\mathcal{A}$, denoted $\mathcal{A}(u) = v$.

2.2 Transition monoid

In order to define a notion of aperiodicity for a transducer, we associate with each two-way automaton a monoid called its *transition monoid*. A transducer is then called *aperiodic* if the transition monoid of its input automaton is aperiodic. Let us recall that a monoid is called aperiodic if it contains no trivial group [2]. Equivalently, a monoid M is aperiodic if there exists a smallest integer n, called the *aperiodicity index*, such that for any element x of M, we have $x^n = x^{n+1}$. Note first that the transition monoid of a transducer is the transition monoid of its input automaton and does not depend of its outputs. Note also that our definition is sound for either deterministic or non-deterministic automata/transducers although we only use it for deterministic ones. Lastly, remark that it extends naturally the notion of transition monoid for one-way automata.

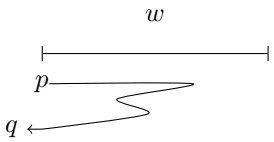


Figure 2 A left-to-left behavior (p, q) of a word w.

The transition monoid is, as usual, obtained by quotienting the free monoid A^* by a congruence which captures the fact that two words have the same *behavior* in the automaton. In an one-way automaton $\mathcal{A}$, the behavior of a word w is the set of pairs (p, q) of states such that there exists a run from p to q in $\mathcal{A}$. Two words are then considered equivalent if their respective behaviors contain the same pairs of states. In a two-way automaton, the behavior of a word is also characterized by the runs it contains but since the reading head can move both ways, the behavior is split into four behaviors called left-to-left, left-to-right, right-to-left and right-to-right behaviors. We only define the left-to-left behavior $\mathrm{bh}_{\ell\ell}(w)$ of a word w. The three other behaviors $\mathrm{bh}_{\ell r}(w)$, $\mathrm{bh}_{r\ell}(w)$ and $\mathrm{bh}_{rr}(w)$ are defined analogously.

Let $\mathcal{A}$ be a two-way automaton. The *left-to-left behavior* $\mathrm{bh}_{\ell\ell}(w)$ of w in $\mathcal{A}$ is the set of pairs (p, q) such that there exists a run which starts at the first position of w in state p and leaves w on the left end in state q (see Figure 2).

Before defining the transition monoid, we illustrate the notion of behavior on the transducer depicted in Figure 1.

▶ **Example 3.** Consider the transducer depicted in Figure 1 and the word $w = aab$. From the run depicted in Figure 1, it can be inferred that

$$\mathrm{bh}_{\ell\ell}(w) = \{(1,2),(2,2)\} \qquad \mathrm{bh}_{r\ell}(w) = \{(1,2)\}$$
$$\mathrm{bh}_{\ell r}(w) = \{(3,1)\} \qquad \mathrm{bh}_{rr}(w) = \{(2,3),(3,1)\}.$$

▶ **Definition 4** (Transition monoid). Let $\mathcal{A} = (Q, A, \delta, q_0, F)$ be a two-way automaton. The transition monoid of $\mathcal{A}$ is $A^*/\sim_{\mathcal{A}}$ where $\sim_{\mathcal{A}}$ is the conjunction of the four relations $\sim_{\ell\ell}$, $\sim_{\ell r}$, $\sim_{r\ell}$ and $\sim_{rr}$ defined for any words w, w' of A^* as follows :

- $w \sim_{\ell\ell} w'$ if $\mathrm{bh}_{\ell\ell}(w) = \mathrm{bh}_{\ell\ell}(w')$.
- $w \sim_{\ell r} w'$ if $\mathrm{bh}_{\ell r}(w) = \mathrm{bh}_{\ell r}(w')$.
- $w \sim_{r\ell} w'$ if $\mathrm{bh}_{r\ell}(w) = \mathrm{bh}_{r\ell}(w')$.
- $w \sim_{rr} w'$ if $\mathrm{bh}_{rr}(w) = \mathrm{bh}_{rr}(w')$.

The neutral element of this monoid is the class of the empty word ϵ, whose behaviors $bh_{xy}(\epsilon)$ is the identity function if $x \neq y$, and is the empty relation otherwise.

These relations are not new and were already evoked in [21, 6] for example. Moreover, the left-to-left behavior was already introduced in [27] to prove the equivalence between one-way and two-way automata.

For a deterministic two-way automaton, the four behaviors $\mathrm{bh}_{\ell\ell}(w)$, $\mathrm{bh}_{\ell r}(w)$, $\mathrm{bh}_{r\ell}(w)$ and $\mathrm{bh}_{rr}(w)$ are partial functions. In the non-deterministic case, these four relations are not functions but relations over the state set Q because there might exist several runs with the same starting state and different ending states. Furthermore, for deterministic automaton, the domains of the functions $\mathrm{bh}_{\ell\ell}(w)$ and $\mathrm{bh}_{\ell r}(w)$ (resp. $\mathrm{bh}_{r\ell}(w)$ and $\mathrm{bh}_{rr}(w)$) are disjoint, since there is a unique run starting in state p at the first (resp. last) position of w. Thus a run starting at the first (resp. last) position leaves w either on the left or the right. For a deterministic two-way automaton, the four behaviors $\mathrm{bh}_{\ell\ell}(w)$, $\mathrm{bh}_{\ell r}(w)$, $\mathrm{bh}_{r\ell}(w)$ and $\mathrm{bh}_{rr}(w)$

$[a] = a^+$
$[b] = b$
$[ab] = a^+b$
$[ba] = ba^+$
$[aba] = aA^*bA^*a$
$[abb] = aA^*bA^*b$
$[bba] = bA^*bA^*a$
$[bb] = bA^*b$

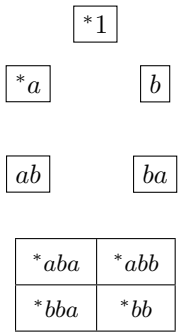


Figure 3 The equivalence classes of the transition monoid and its $\mathcal{D}$-class representation

can be seen as a single partial function f_w from $Q \times \{\ell, r\}$ to $Q \times \{\ell, r\}$ where $f_w(p, x) = (q, y)$ whenever $(p, q) \in \mathrm{bh}_{xy}(w)$ for any $x, y \in \{\ell, r\}$.

▶ **Lemma 5.** *Let $\mathcal{A}$ be a two-way transducer. Then the relation $\sim_{\mathcal{A}}$ is a congruence of finite index.*

It is pure routine to check that $\sim_{\mathcal{A}}$ is indeed a congruence. It is of finite index since each of the four relations $\sim_{\ell\ell}$, $\sim_{\ell r}$, $\sim_{r\ell}$ and $\sim_{rr}$ has at most $2^{|Q|^2}$ classes. Note that the composition of the behaviors is not as straightforward as in the case of one-way automata, the four relations being intertwined. For example, the composition law of the $\mathrm{bh}_{\ell r}$ relation is given by the equality $\mathrm{bh}_{\ell r}(uv) = \mathrm{bh}_{\ell r}(u)\big(\mathrm{bh}_{\ell\ell}(v)\,\mathrm{bh}_{rr}(u)\big)^*\,\mathrm{bh}_{\ell r}(v)$ which follows from the decomposition of a run in uv.

▶ **Example 6.** We illustrate the notion of a transition monoid by giving the one of the transducer depicted in Figure 1. We have omitted all words containing one of the two endmarkers since these words cannot contribute to a group. The eight classes of the congruence $\sim_{\mathcal{A}}$ for the remaining words are given in Figure 3 on the left. The $\mathcal{D}$-class representation of this monoid is also given for the aware reader on the right. It can be checked that this monoid is aperiodic. The transducer of Figure 1 is then aperiodic.

2.3 FO graph transductions

The **MSO**-transductions defined by Courcelle [12] are a variant of the classical logical interpretation of a relational structure into another one. Let us recall that a relational structure S has a $\mathcal{L}$-interpretation, for some logic $\mathcal{L}$, into a structure T if it has an isomorphic copy in T defined by $\mathcal{L}$-formulas. More precisely, this means that there exists a $\mathcal{L}$-formula φ_S with one first-order free variable and a one-to-one correspondence f between the domain of S and the subset T' of elements of T satisfying φ_S. Furthermore, for each relation R of S with arity r, there exists a $\mathcal{L}$-formula φ_R with r first-order free variables such that R is isomorphic via f to the r-tuples of T' satisfying φ_R.

A **MSO**-transduction defines for each input structure a new structure obtained by **MSO**-interpretation into a fixed number of copies of the input structure. In this case, the relations are the letter predicates and the successor relation, which are of arity one and two respectively. To fit into this framework, words are viewed as linear graphs. Each word $w = a_1 \cdots a_n$ is viewed as a linear graph with n vertices carrying the symbols $a_1, \ldots, a_n$. Linear means here that if the vertex set is $\{1, 2, \ldots, n\}$, the edge set is $\{(k, k+1) : 1 \leq k \leq n-1\}$.

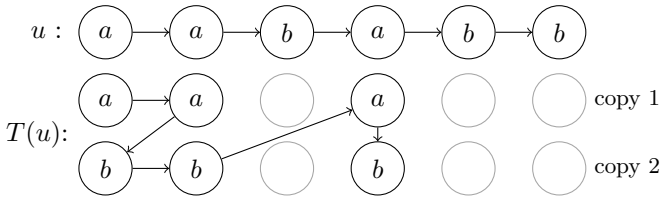


Figure 4 The linear graph of $u = aababb$ and the output structure of T over u.

When restricted to linear graphs, the **MSO**-transductions has been proved to have the same expressive power as two-way transducers [13]. We are interested in this article in **FO** graph transductions, the restriction to first order formulas. Since we consider transductions whose domain is not the set of all graphs, there is an additional closed formula φ_{dom} which determines whether the given graph is in the domain of the transduction.

Before giving the formal definition, we give below an example of a **FO**-transduction. Note that when considering **FO** transductions, the successor relation is replaced by the order relation.

▶ **Example 7.** We give here a **FO** graph transduction that realizes the function f introduced in Example 1. So let $T = (A, A, \varphi_{dom}, C, \varphi_{pos}, \varphi_{\leqslant})$ be the **FO** graph transduction defined as follows :

- $A = \{a, b\}$ is both the input and output alphabet,
- $C = \{1, 2\}$,
- φ_{dom} is a **FO** formula stating that the input is a linear graph,
- $\varphi_a^1(x) = \varphi_b^2(x) = \mathbf{a}(x)$, the other position formulas being set as $false$,
- the order formulas are defined now :
  - $\varphi_{\leqslant}^{i,i}(x, y) = x \leqslant y$ for $i = 1, 2$,
  - $\varphi_{\leqslant}^{1,2}(x, y) = x \leqslant y \vee (\forall z\ y \leqslant z \leqslant x \to \mathbf{a}(z))$,
  - $\varphi_{\leqslant}^{2,1}(x, y) = \exists z\ x \leqslant z \leqslant y \wedge \mathbf{b}(z)$.

▶ **Definition 8.** A **FO**-graph transduction is a tuple $T = (A, B, \varphi_{dom}, C, \varphi_{pos}, \varphi_{\leqslant})$ defined as follows:

- A is the *input alphabet*.
- B is the *output alphabet*.
- φ_{dom} is the *domain formula*. A graph is accepted as input if it satisfies the domain formula.
- C is a finite set, denoting the copies of the input that can exist in the output.
- φ_{pos} is a set of formulas with one free variable $\varphi_b^c(x)$, for $b \in B$ and $c \in C$. Given c, the formulas $\varphi_b^c(x)$, for $b \in B$, are mutually exclusive. The c copy of a node i is labelled by b if, and only if, the formula $\varphi_b^c(x/i)$ is true.
- $\varphi_{\leqslant}$ is a set of formulas with two free variables $\varphi_{\leqslant}^{c,c'}(x, y)$, for $c, c' \in C$. There exists a path from the c copy of a node i to the c' copy of a node j if, and only if, the formula $\varphi_{\leqslant}^{c,c'}(x/i, y/j)$ is true.

All formulas are required to be in **FO**[<] and are evaluated on the input graph.

The output graph is defined as a substructure of the C copies of the input linear graph, in which a node exists if it satisfies one position formula, and is labelled accordingly, and the order is defined according to the order formulas.

In this article, we are only interested in linear graph transductions, which only accept words seen as linear graphs as input. An input word has an *image* by a **FO** graph transduction

if the associated linear graph satisfies its domain formula and the order relation of the output graph, defined by the order formulas, defines a linear graph corresponding to a word. If one condition fails, then the function is undefined on the given input. One should note that the fact that a graph is linear and corresponds to a word is **FO**-definable.

In Figure 4, we give the output structure of T over the linear graph $u = aababb$. Note that for the sake of readability, we do not draw the whole order relation, but simply the successor relation.

3 Main result

We are now ready to state the main result of this article, as an extension of the result by McNaughton and Papert [19] and Schützenberger [25] in the context of two-way transducers and **MSO** transductions established by Engelfriet and Hoogeboom [13].

▶ **Theorem 9.** *The functions realized by aperiodic two-way transducers are exactly the functions realized by* **FO** *graph transductions over words.*

The theorem is proved in Sections 5 and 6. The first inclusion relies on Theorem 13, while the second inclusion stems from the conjunction of Theorems 18, 19 and 20. The next Section is devoted to the composition of transducers, which is a key tool of the proof.

4 Composition of transducers

As transducers realize functions over words, the natural question of the compositionality occurs. In a generic way, this question is : given two functions realized by some machine, can we construct a machine that realizes the composition of these functions. This question has been considered in [16] for generic machines, and resolved positively in the case of deterministic two-way transducers in [10].

This result can also be obtained using the equivalence of two-way transducers with **MSO** transductions, since these are easily proved to be stable by composition (see [12]). However, the reduction from **MSO** transductions to two-way transducers established in [13] makes an extensive use of a weaker version of this result, which is that the composition of a one-way deterministic, called sequential in the following, transducer with a two-way transducer can be done by a two-way transducer, which was first proved in [1].

In this section, we follow this approach, and now prove that this result holds for aperiodic transducers, in the sense that if the two input transducers are aperiodic, then we can construct an aperiodic transducer realizing the composition.

▶ **Theorem 10.** *Let $\mathcal{A}$ be a sequential transducer that can be composed with a two-way transducer $\mathcal{B}$, both deterministic and aperiodic. Then we can effectively construct an aperiodic and deterministic two-way transducer $\mathcal{C}$ such that $\mathcal{C} = \mathcal{B} \circ \mathcal{A}$.*

5 From aperiodic two-way transducers to FO transductions

Let us consider a deterministic and aperiodic two-way transducer. We aim to construct a first-order graph transduction that realizes the same function.

In order to do that, we need to define a formula φ_{dom} for the input domain, formulas φ_{pos} for each copies of a position and each output letter of $\mathcal{A}$, and, contrary to the generic case of **MSO** graph transductions where only the successor is defined, we need here to define

order formulas $\varphi_{\leqslant}$ that describe the order relation on the output depending on the copies of the nodes from the input.

The following result simply stems from the equivalence of aperiodic monoids and first order logic established in [25, 19], but is an essential step to link aperiodicity to first-order, as it is used in the next theorem, which proves that the order relation between positions is first-order definable.

▶ **Lemma 11.** *Let $\mathcal{A} = (Q, A, \delta)$ be an aperiodic two-way automaton. Then the relation classes of $\sim_{\ell\ell}$, $\sim_{\ell r}$, $\sim_{r\ell}$, $\sim_{rr}$ and consequently $\sim_{\mathcal{A}}$ of $\mathcal{A}$ are* **FO***-definable.*

▶ **Lemma 12.** *Let $\mathcal{A}$ be an aperiodic two-way automaton. Then for any pair of states q and q' of $\mathcal{A}$, there exists a* **FO***-formula $\varphi^{q,q'}(x, y)$ such that for any word u in the domain of $\mathcal{A}$ and any pair of positions i and j of u,*

$$u \models \varphi^{q,q'}(x/i, y/j)$$

if, and only if, the run of $\mathcal{A}$ over u starting at position i in state q eventually reaches the position j in state q'.

We now state the main result of this section and construct the first-order transduction that realizes $\mathcal{A}$.

▶ **Theorem 13.** *Let $\mathcal{A}$ be an aperiodic two-way transducer. Then we can effectively construct a* **FO***-graph transduction that realizes the same function as $\mathcal{A}$.*

Proof. For simplicity of the proof, we consider a transducer $\mathcal{A} = (Q, A, B, \delta, \gamma, i, F)$ where the production of any transition is at most one letter. This can be done without loss of generality, since any given transducer can be normalized this way by increasing the number of states. We now give the formal definition of the **FO** transduction $T = (A, B, \varphi_{dom}, Q, \varphi_{pos}, \varphi_{\leqslant})$ that realizes $\mathcal{A}$.

As we consider string transductions within the scope of graph transductions, the domain formula also has to ensure that the input is a linear graph. This can be done in **FO** by a formula stating that there is one position that has no predecessor, one position that has no successor, every other position has exactly one successor and one predecessor and every pair of positions is comparable. Then the domain formula of T is the formula describing the language recognized by the input automaton of $\mathcal{A}$ conjuncted with the linear graph formula. By Lemma 11, as $\mathcal{A}$ is aperiodic the domain formula is **FO**-definable. The order formulas are given by Lemma 12, where obviously $\varphi^{q,q'}_{\leqslant}(x, y) = \varphi^{q,q'}(x, y)$.

The $\varphi^q_b(x)$ formulas, where $q \in Q$ and $b \in B$, express that the production of $\mathcal{A}$ at the position quantified by x in state q is b, but also that the run of $\mathcal{A}$ over u reaches the said position in state q. Should we define $A_{b,q} = \{a \in A \mid \gamma(a, q) = b\}$, then the first condition is expressed as $\bigvee_{a \in A_{b,q}} a(x)$. The second condition is then equivalent to saying that there exists a run from the initial state of $\mathcal{A}$ to the current position, which is expressed by the formula $\exists y \forall z\ y \leqslant z \wedge \varphi^{i,q}(y, x)$. The formula $\varphi^q_b(x)$ is thus defined as the conjunction of these two formulas.

The transduction T is now defined. All formulas are expressed in the first order logic, and it realizes the same function as $\mathcal{A}$, proving the theorem. ◀

6 From FO transductions to aperiodic two-way transducers

The proof scheme for this inclusion is adapted from the one in [13] proving that **MSO** transductions are realized by two-way deterministic transducers. We prove that we can

construct an aperiodic two-way transducer with **FO** look around from a **FO** transduction, and that the constructions given in [13] suppressing the look around part preserve the aperiodicity.

We define in the next subsection the models of transducers with look-around that are used in the proof. We then give an alternative definition of aperiodicity which can be applied to transducers with logic look-around before explaining the constructions that lead up to the result.

6.1 Transducers with look-around

Here, we define two kinds of transducers with look-around. The first one is a restriction of two-way transducers with regular look-around, where we limit the regular languages used in the tests to Star-free languages, which is the rational characterization of first-order logic. These transducers differ from the classic ones by their transitions, where the tests are not determined by the letter read, but also by the prefix and suffix which can be evaluated according to some regular languages.

The second extension we consider is transducers with first-order look around. In this case, the selection of a transition, as well as the movements of the reading head, are determined by formulas. Formal definitions are given below.

In both cases only the definition of transition is changed, the definition of run and accepting run remaining the same.

▶ **Definition 14** (two-way transducer with Star-Free look around)**.** *Two-way transducers with Star-Free look around* are a subclass of two-way transducers with regular look around defined in [13], where all languages in the tests are Star-Free.

Formally, it is a machine $\mathcal{A} = (Q, A, B, \Delta, i, F)$ where Q, A, B, i and F are the same as for two-way transducers, and transitions and productions are regrouped in Δ, and are of the form (q, t, q', v, m) where q and q' are states from Q, $v \in B^*$ is the production of the transition, $m \in \{-1, 0, +1\}$ describes the movement of the reading head and t is a test of the form (L_p, a, L_s) where a is a letter of $A \uplus \{\vdash, \dashv\}$, and L_p and L_s are Star-Free languages over the same alphabet. A test (L_p, a, L_s) is satisfied if the reading head is on a position labelled by the letter a, the prefix of the input word up to the position of the reading head belongs to L_p, and symmetrically the suffix belongs to L_s.

Such a machine is deterministic if the tests performed in a given state are mutually exclusives.

▶ **Definition 15** (two-way transducer with **FO** look around)**.** *Two-way transducers with* **FO** *look around* are a subclass of two-way transducers with **MSO** look around where formulas are restricted to the first-order.

Formally, it is a machine $\mathcal{A} = (Q, A, B, \Delta, i, F)$ where Q, A, B, i and F are the same as two-way transducers, and transitions of Δ are of the form $(q, \varphi(x), q', v, \psi(x, y))$ where q and q' are states from Q, $v \in B^*$ is the production of the transition and $\varphi(x)$ and $\psi(x, y)$ are **FO** formulas with respectively one and two free variables. A transition $(q, \varphi(x), q', v, \psi(x, y))$ can be taken if the formula $\varphi(x)$ holds on the input word, where x quantifies the current position i of the reading head, say $\vdash u \dashv \models \varphi(x/i)$. Then the reading head moves to a position j such that $\vdash u \dashv \models \psi(x/i, y/j)$.

Such a machine is deterministic if the unary tests appearing in a given state are mutually exclusive, and if for any input word u, any movement formula $\psi(x, y)$ and any position i, there exists at most one position j such that $\vdash u \dashv \models \psi(x/i, y/j)$.

6.2 Aperiodicity by path contexts

The reading head of transducers with logic look-around can jump several positions at a time and in any direction. Then the notion of behavior for such transducers becomes blurry, since behaviors would have to be considered starting at any position, and moreover the direction taken while exiting a word is not decided locally, but depends on the context.

We thus give an equivalent characterization of the aperiodicity of a transducer through all contexts at a time, for machines whose reading head does not move position by position.

We recall that given a (deterministic) transducer $\mathcal{A} = (Q, A, B, \Delta, init, F)$ and u an input word of $\mathcal{A}$, the accepting path of $\mathcal{A}$ over u, denoted $path(u)$, is the sequence $(q_0, i_0) \dots (q_n, i_n)$ of pairs from $Q \times [0, |u| + 1]$ (the length of u plus the endmarkers) describing the behavior of the reading head of $\mathcal{A}$ while reading u, as defined in Subsection 2.1.

We now define the projection of paths, as a way to highlight some information and forget the rest. It is applied to contexts in order to only retain the influence of a word on its context.

▶ **Definition 16** (Projection and context paths)**.** Let $I = [i_1, \dots, i_k]$ be an ordered sequence of integers. We define $path_I(u)$ as the sequence of pairs from $Q \times \{1, \dots, k\}$ such that for any pairs (q, j) and (q', j'), (q, j) appears before (q', j') in $path_I(u)$ if, and only if, (q, i_j) appears before $(q', i_{j'})$ in $path(u)$. Informally, this corresponds to selecting pairs whose position is in I and renaming them according to the set I.

Abusing notations, we will note the *context path* $path_{vw}(vuw) = path_I(vuw)$ where I is the set of positions of v and w.

Then $path_{vw}(vuw)$ is the trace of the run over u on the context v, w and two words u and u' are $\mathcal{A}$-equivalent if for any context v, w, we have equality of the paths contexts $path_{vw}(vuw) = path_{vw}(vu'w)$. Then by definition of the aperiodicity, a transducer or an automaton $\mathcal{A}$ is *aperiodic* if there exists a positive integer n such that for any words u, v and w on the input alphabet of $\mathcal{A}$, the context paths $path_{vw}(vu^n w)$ and $path_{vw}(vu^{n+1}w)$ are equals. One should remark that on two-way transducers, this notion is equivalent to the aperiodicity of the transition monoid. The next lemma serves as the link from the first-order logic to the aperiodicity by context paths.

▶ **Lemma 17.** *Let T be a* **FO** *graph transduction. There exists a positive integer n such that for any input words u, v and w such that $vu^n w$ is in the domain of T, $vu^{n+1}w$ is also in the domain of T and the two words satisfy the same formulas of T, when the free variables quantify positions of v or w.*

Proof. First consider the domain formula of T. Since it is a **FO** formula, it has an aperiodicity index n, in the sense that for any words u, v and w, $vu^n w$ is in the domain of T if, and only if, $vu^{n+1}w$ is in the domain of T.

We now prove the result in the case where i ranges over v and j ranges over w, but similar proofs hold for i and j ranging independently over v and w. Consider a pair c, c' of copies in C, and integers $0 \leqslant i < |v|$ and $0 \leqslant j < |w|$. Then a word with positions i and j quantified respectively by x and y can be seen as a word over the alphabet $A \times \{0, 1\}^2$, where all letters have $(0, 0)$ as second component, except $(v_i, 1, 0)$ and $(w_j, 0, 1)$. The formula $\varphi_{\leqslant}^{c,c'}(x, y)$ can then be equivalently seen as a closed formula over this enriched alphabet. This formula being in **FO**, it describes an aperiodic language, and then there exists an integer n' such that $vu^{n'}w$ satisfies $\varphi_{\leqslant}^{c,c'}(x/i, y/|vu^{n'}| + j)$ if, and only if, $vu^{n'+1}w$ satisfies $\varphi_{\leqslant}^{c,c'}(x/i, y/|vu^{n'+1}| + j)$.

A similar argument also holds for the node formulas $\varphi_b^c(x)$. As there is a finite number of formulas, there exists an integer, the maximum of the index of each formulas, such that the result holds. ◀

This lemma means that the transducer has an aperiodicity index, in the sense that u^n and u^{n+1} behave the same way for the same context. It also corresponds to the notion of aperiodicity defined earlier in this section, where the sequence ranges over pairs of copy and position.

6.3 Construction of the aperiodic transducer

We now hold all the necessary tools to prove the reduction from **FO** transductions to aperiodic two-way transducers.

We present the first construction, from a **FO** transduction to a two-way transducer with **FO** look around. The construction is quite simple. By putting the copy set of the graph transduction as the set of states of the transducer, we can use the fact that the reading head of a transducer with logic look around jumps between positions to strictly follow the output structure of the input transduction. We then use Lemma 17 to prove the aperiodicity of the construction.

▶ **Theorem 18.** *Let T be a* **FO** *graph transduction. Then we can effectively construct an aperiodic two-way transducer with* **FO** *look around that realizes the same function over words.*

We have now constructed an aperiodic two-way machine from the input **FO** transduction. But even though two-way transducers with **MSO** look around are known to be equivalent to two-way transducers [13], we need to prove that we can suppress the **FO** look around while preserving the aperiodicity of the construction. This is done by the two following theorems, using Star-free look around as an intermediate step. We show that the construction evoked in [13] do preserve the aperiodicity, leading to the result.

▶ **Theorem 19.** *Given an aperiodic two-way transducers with* **FO** *look around, we can construct an aperiodic two-way transducers with Star-Free look around that realizes the same function.*

Proof. In order to prove this theorem, we rely on the proof of Lemma 6 from [13], which proves that two-way transducers with **MSO** look around can be expressed by two-way transducers with regular look around. This is done by constructing a transducer whose regular tests stem directly from the **MSO** formulas. Then the reading head simulates the jumps of the reading head of the transducer with **MSO** look around by moving step by step up to the required position.

We aim to prove on one hand that if the formulas are defined in the first-order, then the resulting two-way transducer only uses Star-free look around, and on the other hand that if moreover the input transducer is aperiodic, then the output transducer is also aperiodic.

The first claim is proved by noticing that the languages used in the regular look around construction are languages defined by formulas of the input transducer with **FO** look around with free variables. Then if the free variables are seen as an enrichment of the alphabet, similarly to what is done in the proof of Lemma 17, the formula remains first-order, and consequently all the languages used in look around tests are Star-free.

Now let us compare the moves of the reading head of the resulting transducer with Star-free look around with the ones of the head of the input transducer with **FO** look around. The path of the resulting transducer over any word can entirely be deduced from the path of the input transducer, by adding step by step walks between the jumps of the reading head. Then, if the input transducer is aperiodic with index n, given three words u, v and w, the context paths $path_{vw}(vu^n w)$ and $path_{vw}(vu^{n+1}w)$ are equal, and thus the context paths for

the transducer with Star-free look around, that are deduced from it, are equal too, proving the aperiodicity of the resulting transducer. ◀

We finally prove that we can suppress the Star-free look around tests while preserving the aperiodicity, which concludes the proof of the main theorem.

▶ **Theorem 20.** *Given an aperiodic two-way with Star-free look around, we can construct an aperiodic two-way transducers that realizes the same function.*

Proof. Here again, we consider the construction from [13], Lemma 4, proving that we can suppress the regular look around tests. Our goal is then to prove that this operation preserves the aperiodicity when the input transducer only uses Star-free languages.

The construction relies heavily on the fact that the composition of a two-way transducer with a one-way transducer can be done by a two-way transducer. It is used to preprocess the input by adding the result of each regular tests from the transitions at each position. Given the test (L_p, a, L_s) of a transition, a left-to-right pass simulates L_p and reproduces the input where each position is enriched with the information : does its prefix belong to L_p. Symmetrically, a right-to-left transducer adds the same information for L_s. Let us remark that since these languages are Star-free, the input automaton of the transducers simulating these languages are aperiodic.

Then the information regarding every transition is added to the input, and lastly we can construct a two-way transducer that acts in the same way as the input transducer, but where all the look around have been suppressed and are done locally by looking at the enrichment part of the letter. This two-way transducer without look around is aperiodic if the input transducer is aperiodic, since they share the same paths, and thus context paths.

Finally, the input transducer is given as the composition of a single aperiodic two-way transducer with a finite number of aperiodic one-way transducers. Should we first remark that, by symmetry of the problem, Theorem 10 also holds for right sequential transducers, through several uses of this composition result we finally obtain a unique two-way transducer that realizes the input transducer with Star-free look around. ◀

7 Conclusion

We recall that a similar work has been done for streaming string transducers by Filiot, Krishna and Trivedi [14]. Then through **FO** transductions, this result and Theorem 9 prove the equivalence of aperiodicity for the two models of transducers.

There exists algorithms that input a two-way transducer and construct directly an equivalent streaming string transducer (see [4] for example). It would be interesting to check first if the aperiodicity is preserved through these algorithms, and secondly to compare the size and aperiodicity indexes of the two transition monoids. Although unknown from the authors, a reciprocal procedure and its study would hold the same interest.

On a more generic note, one can ask which fragments of logic preserve their algebraic characterization in the scope of two-way transducers and **MSO** transductions. For example, are $\mathcal{J}$-trivial transducers equivalent to $\mathcal{B}\Sigma_1$ transductions ? The main challenges for this question are the stability by composition of these restricted classes of transducers on one hand, and on the other hand the very definition of logic transductions for restricted fragments, as a fragment must retain some fundamental expressive properties, such as being able to characterize linear graphs.

Finally, we would like to point out the fact that even if we can decide if a given two-way transducer is aperiodic, it is still open to decide if the function realized by a two-way

transducer can be realized by an aperiodic one. An promising approach for this problem might be to consider machine-independent descriptions of functions, as defined recently for streaming string transducers in [5] for example. This was successfully done in [7] for machines with origin semantic. We also think that this question could be solved by the notion of canonical object of a function over words, which has yet to be defined.

Acknowledgements. We would like to thank Antoine Durand-Gasselin, Pierre-Alain Reynier and Jean-Marc Talbot for very fruitful discussions.

References

1. A. V. Aho, J. E. Hopcroft, and J. D. Ullman. A general theory of translation. *Math. Systems Theory*, 3:193–221, 1969.
2. Jorge Almeida. *Finite semigroups and universal algebra*, volume 3 of *Series in Algebra*. World Scientific Publishing Co., Inc., River Edge, NJ, 1994. Translated from the 1992 Portuguese original and revised by the author.
3. Rajeev Alur and Pavol Černý. Expressiveness of streaming string transducers. In *30th International Conference on Foundations of Software Technology and Theoretical Computer Science (STACS'10)*, volume 8 of *LIPIcs – Leibniz International Proceedings in Informatics*, pages 1–12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Wadern, 2010.
4. Rajeev Alur, Emmanuel Filiot, and Ashutosh Trivedi. Regular transformations of infinite strings. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*, pages 65–74. IEEE Computer Society, 2012.
5. Rajeev Alur, Adam Freilich, and Mukund Raghothaman. Regular combinators for string transformations. In Thomas A. Henzinger and Dale Miller, editors, *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS'14, Vienna, Austria, July 14–18, 2014*, page 9. ACM, 2014.
6. Jean-Camille Birget. Concatenation of inputs in a two-way automaton. *Theoret. Comput. Sci.*, 63(2):141–156, 1989.
7. Mikolaj Bojanczyk. Transducers with origin information. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming – 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8–11, 2014, Proceedings, Part II*, volume 8573 of *Lecture Notes in Computer Science*, pages 26–37. Springer, 2014.
8. J. Richard Büchi. Weak second-order arithmetic and finite automata. *Z. Math. Logik Grundlagen Math.*, 6:66–92, 1960.
9. Christian Choffrut. Une caractérisation des fonctions séquentielles et des fonctions sous-séquentielles en tant que relations rationnelles. *Theoret. Comput. Sci.*, 5:325–338, 1977.
10. Michal P. Chytil and Vojtěch Jákl. Serial composition of 2-way finite-state transducers and simple programs on strings. In *Automata, languages and programming (Fourth Colloq., Univ. Turku, Turku, 1977)*, pages 135–137. Lecture Notes in Comput. Sci., Vol. 52. Springer, Berlin, 1977.
11. A. Cohen and J.-F. Collard. Instance-wise reaching definition analysis for recursive programs using context-free transductions. In *PACT'98*, 1998.
12. Bruno Courcelle. Monadic second-order definable graph transductions: a survey [see MR1251992 (94f:68009)]. *Theoret. Comput. Sci.*, 126(1):53–75, 1994. Seventeenth Colloquium on Trees in Algebra and Programming (CAAP'92) and European Symposium on Programming (ESOP) (Rennes, 1992).

13 Joost Engelfriet and Hendrik Jan Hoogeboom. MSO definable string transductions and two-way finite-state transducers. *ACM Trans. Comput. Log.*, 2(2):216–254, 2001.

14 Emmanuel Filiot, Shankara Narayanan Krishna, and Ashutosh Trivedi. First-order definable string transformations. In Venkatesh Raman and S. P. Suresh, editors, *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS'14, December 15–17, 2014, New Delhi, India*, volume 29 of *LIPIcs – Leibniz International Proceedings in Informatics*, pages 147–159. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Wadern, 2014.

15 Christiane Frougny. Numeration systems. In M. Lothaire, editor, *Algebraic Combinatorics on Words*. Cambridge, 1999. to appear.

16 J. E. Hopcroft and J. D. Ullman. An approach to a unified theory of automata. *Bell System Tech. J.*, 46:1793–1829, 1967.

17 S. C. Kleene. Representation of events in nerve nets and finite automata. In *Automata studies*, Annals of mathematics studies, no. 34, pages 3–41. Princeton University Press, Princeton, N. J., 1956.

18 Pierre McKenzie, Thomas Schwentick, Denis Thérien, and Heribert Vollmer. The many faces of a translation. In *Automata, languages and programming (Geneva, 2000)*, volume 1853 of *Lecture Notes in Comput. Sci.*, pages 890–901. Springer, Berlin, 2000.

19 Robert McNaughton and Seymour Papert. *Counter-free automata*. The M.I.T. Press, Cambridge, Mass.-London, 1971.

20 A. Nerode. Linear automaton transformation. In *Proceeding of the AMS*, volume 9, pages 541–544, 1958.

21 J.-P. Pécuchet. Automates boustrophédon, semi-groupe de Birget et monoïde inversif libre. *RAIRO Inform. Théor.*, 19(1):71–100, 1985.

22 M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3, 1959.

23 Christophe Reutenauer and Marcel-Paul Schützenberger. Variétés et fonctions rationnelles. *Theoretical Computer Science*, 145:229–240, 1995.

24 Emmanuel Roche and Yves Schabes. *Finite-State Language Processing*, chapter 7. MIT Press, Cambridge, 1997.

25 Marcel-Paul Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8:190–194, 1965.

26 Marcel-Paul Schützenberger. Sur le produit de concaténation non ambigu. *Semigroup Forum*, 13(1):47–75, 1976/77.

27 J. C. Shepherdson. The reduction of two-way automata to one-way automata. *IBM J. Res. Develop.*, 3:198–200, 1959.

28 Denis Thérien and Thomas Wilke. Over words, two variables are as powerful as one quantifier alternation. In *ACM STOCS'98*, pages 256–263, 1998.

On Relative and Probabilistic Finite Counterability

Orna Kupferman and Gal Vardi

School of Engineering and Computer Science, Hebrew University, Israel

Abstract

A *counterexample* to the satisfaction of a linear property ψ in a system $\mathcal{S}$ is an infinite computation of $\mathcal{S}$ that violates ψ. When ψ is a safety property, a counterexample to its satisfaction need not be infinite. Rather, it is a *bad-prefix* for ψ: a finite word all whose extensions violate ψ. The existence of finite counterexamples is very helpful in practice. Liveness properties do not have bad-prefixes and thus do not have finite counterexamples.

We extend the notion of finite counterexamples to non-safety properties. We study *counterable languages* – ones that have at least one bad-prefix. Thus, a language is counterable iff it is not liveness. Three natural problems arise: (1) Given a language, decide whether it is counterable, (2) study the length of minimal bad-prefixes for counterable languages, and (3) develop algorithms for detecting bad-prefixes for counterable languages. We solve the problems for languages given by means of LTL formulas or nondeterministic Büchi automata. In particular, our EXPSPACE-completeness proof for the problem of deciding whether a given LTL formula is counterable, and hence also for deciding liveness, settles a long-standing open problem.

In addition, we make finite counterexamples more relevant and helpful by introducing two variants of the traditional definition of bad-prefixes. The first adds a *probabilistic* component to the definition. There, a prefix is bad if almost all its extensions violate the property. The second makes it *relative* to the system. There, a prefix is bad if all its extensions in the system violate the property. We also study the combination of the probabilistic and relative variants. Our framework suggests new variants also for safety and liveness languages. We solve the above three problems for the different variants. Interestingly, the probabilistic variant not only increases the chances to return finite counterexamples, but also makes the solution of the three problems exponentially easier.

1998 ACM Subject Classification F.4.3 Formal Languages, B.8.2 Performance Analysis and Design Aids, F.1.1 Models of Computation

Keywords and phrases Model Checking, Counterexamples, Safety, Liveness, Probability

Digital Object Identifier 10.4230/LIPIcs.CSL.2015.175

1 Introduction

In model checking, we verify that a system meets a desired property by checking that a mathematical model of the system meets a formal specification that describes the property. *Safety* and *liveness* [2] are two classes of system properties. Essentially, a safety property states that something "bad" never happens and a liveness property states that something "good" eventually happens. Formally, consider a language L of infinite words over an alphabet Σ. A finite word $x \in \Sigma^*$ is a *bad-prefix* for L if for all infinite words $y \in \Sigma^\omega$, the concatenation $x \cdot y$ is not in L. Thus, a bad-prefix for L is a finite word that cannot be extended to a word in L. A language L is *safety* if every word not in L has a bad-prefix, and is *liveness* if it has no bad-prefixes. Thus, every word in Σ^* can be extended to a word in L.

The classes of safety and liveness properties have been extensively studied. From a theoretical point of view, their importance stems from their topological characteristics. Consider the natural topology on Σ^ω, where similarity between words corresponds to the

length of the prefix they share. Formally, the distance between w and w' is 2^{-i}, where $i \geq 0$ is the position of the first letter in which w and w' differ. In this topology, safety and liveness properties are exactly the closed and dense sets, respectively [2]. This, for example, implies that every linear property can be expressed as a conjunction of a safety and a liveness property [2, 3].

From a practical point of view, reasoning about safety and liveness properties require different methodologies, and the distinction between them pinpoints different challenges of formal methods. In liveness, one has to demonstrate progress towards fulfilling eventualities. Thus, liveness is the concept behind fairness [15], and behind the need for rich ω-regular acceptance conditions [37], progress measures [30, 27], and many more. On the other hand, in safety one can reason about finite computations of the system [35]. The latter has significant advantages in symbolic methods [23], bounded model checking [7], run-time verification [17], synthesis [25], and many more.

An important advantage of model-checking tools is their ability to accompany a negative answer to the correctness query by a *counterexample* to the satisfaction of the specification in the system. Thus, together with a negative answer, the model checker returns some erroneous execution of the system. These counterexamples are not only essential in detecting subtle errors in complex systems [9], but also in improving the modeling of systems. For example, in CEGAR, counterexamples are used in order to guide the refinement of abstract models [8]. In the general case, the erroneous execution of the system is infinite. It is known, however, that for linear temporal logic (LTL) properties, there is always a *lasso-shaped* counterexample – one of the form uv^ω, for finite computations u and v. Clearly, the simpler the counterexample is, the more helpful it is for the user, and indeed there have been efforts for designing algorithms that return short counterexamples [33, 22]. The analysis of counterexamples makes safety properties even more appealing: rather than a lasso-shaped counterexample, it is possible to return to the user a bad-prefix. This enables the user to find errors as soon as they appear. In addition, the analysis of bad-prefixes is often more helpful, as they point the user not just to one erroneous execution, but rather to a finite execution all whose continuations are erroneous.

We extend the notion of finite counterexamples to non-safety specifications. We also make finite counterexamples more relevant and helpful by introducing two variants of the traditional definition of bad-prefixes. The first adds a *probabilistic* component to the definition. The second makes it *relative* to the system. We also consider the combination of the probabilistic and relative variants. Before we describe our contribution in detail, let us demonstrate the idea with the following example. Consider a system $\mathcal{S}$ and a specification ψ stating that every request is eventually followed by a response. There might be some input sequence that leads $\mathcal{S}$ to an error state in which it stops sending responses. While ψ is not safety, the system $\mathcal{S}$ has a computation with a prefix that is bad with respect to $\mathcal{S}$: all its extensions in $\mathcal{S}$ do not satisfy ψ. Returning this prefix to the user, with its identification as bad with respect to $\mathcal{S}$, is more helpful than returning a lasso-shaped counterexample. Consider now a specification φ stating that the system eventually stops allocating memory. There might be some input sequence that leads $\mathcal{S}$ to a state in which every request is followed by a memory allocation. A computation that reaches this state almost surely violates the specification. Indeed, it is possible that requests eventually stop arriving and the specification would be satisfied, but the probability of this behavior of the input is 0. Thus, the system $\mathcal{S}$ has a computation with a prefix that is bad with respect to $\mathcal{S}$ in a probabilistic sense: almost all of its extensions in $\mathcal{S}$ do not satisfy φ. Again, we want to return this prefix to the user, identified as bad with high probability.

Recall that a language L is liveness if every finite word can be extended to an infinite word in L. Equivalently, L has no bad-prefixes. We say that L is *counterable* if it has a bad-prefix. That is, L is counterable iff it is not liveness. Note that a language, for example $a^* \cdot b \cdot (a + b + c)^\omega$, may be counterable and not safety. When a system does not satisfy a counterable specification ψ, it may contain a bad-prefix for ψ, which we would like to return to the user. Three natural problems arise: (1) Given a language, decide whether it is counterable, (2) study the length of minimal bad-prefixes for counterable languages, and (3) develop algorithms for detecting bad-prefixes for counterable languages.

In fact, the last two problems are open also for safety languages. Deciding whether a given language is safety is known to be PSPACE-complete for languages given by LTL formulas or nondeterministic Büchi word automata (NBWs, for short). For the problem of deciding whether a language is counterable, an EXPSPACE upper-bound for languages given by LTL formulas is not difficult [35], yet the tight complexity is open. This is surprising, as recall that a language is counterable iff it is not liveness, and one could expect the complexity of deciding liveness to be settled by now. As it turns out, the problem was studied in [29], where it is stated to be PSPACE-complete. The proof in [29], however, is not convincing, and indeed efforts to solve the problem have continued, and the problem was declared open in [4] (see also [26]). Our first contribution is an EXPSPACE lower bound, implying that the long-standing open problem of deciding liveness (and hence, also counterability) of a given LTL formula is EXPSPACE-complete. In a recent communication with Diekert, Muscholl, and Walukiewicz, we have learned that they recently came-up with an independent EXPSPACE lower bound, in the context of monitoring or infinite computations [14]. For languages given by means of an NBW, the problem is PSPACE-complete [35, 29]. Thus, interestingly, while in deciding safety the exponential succinctness of LTL with respect to NBWs does not make the problem more complex, in deciding liveness it makes the problem exponentially more complex. This phenomenon is reflected also in the solutions to the problems about the length and the detection of bad-prefixes. We also show that when a language given by an LTL formula is safety, the solutions for the three problems become exponentially easier.

Let us return to our primary interest, of finding finite counterexamples.

Consider a system modelled by a Kripke structure K over a set AP of atomic propositions. Let $\Sigma = 2^{AP}$, and consider an ω-regular language $L \subseteq \Sigma^\omega$. We say that a finite computation $x \in \Sigma^*$ of K is a *K-bad-prefix*, if x cannot be extended to an infinite computation of K that is in L. Formally, for all $y \in \Sigma^\omega$, if $x \cdot y$ is a computation of K, then it is not in L. Once we define K-bad-prefixes, the definitions of safety and counterability are naturally extended to the relative setting: A language L is K-counterable if it has a K-bad-prefix and is K-safety if every computation of K that is not in L has a K-bad-prefix. Using a product of K with an NBW for L, we are able to show that the solutions we suggest for the three problems in the non-relative setting apply also to the relative one, with an additional NLOGSPACE or linear-time dependency in the size of K. We also study K-safety, and the case L is K-safety. We note that relative bad prefixes have already been considered in the literature, with different motivation and results. In [18], where the notion is explicitly defined, as well as in [5, 34], where it is implicit, the goal is to lift the practical advantages of safety to liveness properties, typically by taking the finiteness of the system into an account. In [29], the idea is to relay on fairness conditions known about the system in order to simplify the reasoning about liveness properties, especially in a setting where an abstract model of the system is used.

We continue to the probabilistic view.[1] A random word over Σ is a word in which all letters are drawn from Σ uniformly at random.[2] In particular, when $\Sigma = 2^{AP}$, then the probability of each atomic proposition to hold in each position is $\frac{1}{2}$. Consider a language $L \subseteq \Sigma^\omega$. We say that a finite word $x \in \Sigma^*$ is a *prob-bad-prefix* for L if the probability of an infinite word with prefix x to be in L is 0. Formally, $Pr(\{y \in \Sigma^\omega : x \cdot y \in L\}) = 0$. Then, L is *prob-counterable* if it has a prob-bad-prefix. Now, given a Kripke structure K, we combine the relative and probabilistic views in the expected way: a finite computation $x \in (2^{AP})^*$ of K is a *K-prob-bad-prefix* for L if a computation of K obtained by continuing x with some random walk on K, is almost surely not in L. Thus, a computation of K that starts with x and continues according to some random walk on K is in L with probability 0. We show that this definition is independent of the probabilities of the transitions in the random walk on K. Again, L is *K-prob-counterable* if it has a K-prob-bad-prefix.

We note that a different approach to probabilistic counterexamples is taken in [1]. There, the focus is on reachability properties, namely properties of the form "the probability of reaching a set T of states starting from state s is at most p". Accordingly, a counterexample is a set of paths from s to T, such that the probability of the event of taking some path in the set is greater than p. We, on the other hand, cover all ω-regular languages, and a counterexample is a single finite path – one whose extension result in a counterexample in high probability.

We study the theoretical properties of the probabilistic setting and show that an ω-regular language L is prob-counterable iff the probability of a random word to be in L is less than 1. We also show that ω-regular languages have a "safety-like" behavior in the sense that the probability of a word not to be in L and not to have a prob-bad-prefix is 0. Similar properties hold in the relative setting and suggest that attempts to return to the user prob-bad-prefixes and K-prob-bad-prefixes are likely to succeed.

From a practical point of view, we show that the probabilistic setting not only increases our chances to return finite counterexamples, but also makes the solution of our three basic problems easier: deciding prob-counterability and K-prob-counterability for LTL formulas is exponentially easier than deciding counterability and K-counterability! Moreover, the length of bad-prefixes is exponentially smaller, and finding them is exponentially easier. Our results involve a careful analysis of the product of K with an automaton for L. Now, the product is defined as a Markov chain, and we also need the automaton to be deterministic. Our construction also suggest a simpler proof to the known probabilistic NBW model-checking result of [11]. While the blow-up determinization involves is legitimate in the case L is given by an NBW, it leads to a doubly-exponential blow-up in the case L is given by an LTL formula ψ. We show that in this case, we can avoid the construction of a product Markov chain and, adopting an idea from [11], generate instead a sequence of Markov chains, each obtained from its predecessor by refining the states according to the probability of the innermost temporal subformula of ψ.

It is easy to see that there is a trade-off between the length of a counterexample and its "precision", in the sense that the longer a finite prefix of an erroneous computation is, the larger is the probability in which it is a K-prob-bad-prefix. We allow the user to play with this trade-off and study both the problem in which the user provides, in addition to K and

[1] In [19], the authors study safety and liveness in probabilistic systems. The setting, definitions, and goals are different from ours here, and focus on the safety and liveness fragments of the probabilistic branching-time logic PCTL.

[2] Our definitions and results apply for all distributions in which all letters are drawn with a positive probability.

ψ, also a probability $0 < \gamma < 1$, and gets back a shortest finite computation x of K such that the probability of a computation of K that starts with x to satisfy ψ is less than γ, and the problem in which the user provides a length $m \geq 1$ and gets back a finite computation x of K of length at most m such that the probability of a computation of K that starts with x to satisfy ψ is minimal.

Due to lack of space, detailed proofs can be found in the full version, in the authors' home pages.

2 Preliminaries

2.1 Automata and LTL

A *nondeterministic automaton on infinite words* is a tuple $\mathcal{A} = \langle \Sigma, Q, Q_0, \delta, \alpha \rangle$, where Q is a set of states, $Q_0 \subseteq Q$ is a set of initial states, $\delta : Q \times \Sigma \to 2^Q$ is a transition function, and α is an acceptance condition whose type depends on the class of $\mathcal{A}$. A run of $\mathcal{A}$ on a word $w = \sigma_0 \cdot \sigma_1 \cdots \in \Sigma^\omega$ is a sequence of states $r = q_0, q_1, \ldots$ such that $q_0 \in Q_0$ and $q_{i+1} \in \delta(q_i, \sigma_i)$ for all $i \geq 0$. The run is accepting if it satisfies the condition α. We consider here *Büchi* and *parity* automata. In a Büchi automaton, $\alpha \subseteq Q$ and the run r satisfies α if it visits some state in α infinitely often. Formally, let $inf(r) = \{q : q = q_i \text{ for infinitely many } i\text{'s}\}$ be the set of states that r visits infinitely often. Then, r satisfies α iff $inf(r) \cap \alpha \neq \emptyset$ [6]. In a parity automaton, $\alpha : Q \to \{0, \ldots, k\}$ maps each state to a color in $\{0, \ldots, k\}$. A run r satisfies α if the minimal color that is visited infinitely often is even. Formally, the minimal color c such that $inf(r) \cap \alpha^{-1}(c) \neq \emptyset$ is even. A word $w \in \Sigma^\omega$ is *accepted* by $\mathcal{A}$ if there is an accepting run of $\mathcal{A}$ on w. The *language* of $\mathcal{A}$, denoted $L(\mathcal{A})$, is the set of words that $\mathcal{A}$ accepts. When $|Q_0| = 1$ and $|\delta(q, \sigma)| = 1$ for all $q \in Q$ and $\sigma \in \Sigma$, then $\mathcal{A}$ is *deterministic*. When a state $q \in Q$ is such that no word is accepted from q (equivalently, the language of $\mathcal{A}$ with initial state q is empty), we say that q is *empty*. We use the acronyms NBW, DBW, NPW, and DPW to denote nondeterministic and deterministic Büchi and parity word automata, respectively. We also refer to the standard nondeterministic and deterministic automaton on finite words, abbreviated NFW and DFW, respectively. We define the size of $\mathcal{A}$, denoted $|\mathcal{A}|$, as the size of δ.

An automaton $\mathcal{A}$ induces a graph $G_\mathcal{A} = \langle Q, E \rangle$ where $(q, q') \in E$ iff there is $\sigma \in \Sigma$ such that $q' \in \delta(q, \sigma)$. When we refer to the *strongly connected sets* (SCSs) of $\mathcal{A}$, we refer to the SCSs of this graph. Formally, a set $C \subseteq Q$ of states is an SCS of $\mathcal{A}$ if for all $q, q' \in C$, there is a path from q to q' in $G_\mathcal{A}$. An SCS C is *maximal* if for all sets C' such that $C' \not\subseteq C$, the set $C \cup C'$ is no longer an SCS. A maximal SCS is termed a *strongly connected component* (SCC). An SCC C is *accepting* if a run that visits exactly all the states in C infinitely often satisfies α. For example, when α is a parity condition, then C is accepting if the minimal color c such that $C \cap \alpha^{-1}(c) \neq \emptyset$ is even. An SCC C is *ergodic* iff for all $(q, q') \in E$, if $q \in C$ then $q' \in C$. That is, an SCC is ergodic if no edge leaves it.

The logic *LTL* is a linear temporal logic [32]. Formulas of LTL are constructed from a set AP of atomic proposition using the usual Boolean operators and the temporal operators G ("always"), F ("eventually"), X ("next time"), and U ("until"). The semantics of LTL is defined with respect to infinite computations over AP. We use $w \models \psi$ to indicate that the computation $w \in (2^{AP})^\omega$ satisfies the LTL formula ψ. The language of an LTL formula ψ, denoted $L(\psi)$, is the set of infinite computations that satisfy ψ. For the full syntax and semantics of LTL see [32]. We define the size of an LTL formula ψ, denoted $|\psi|$, to be the number of its Boolean and temporal operators. Given an LTL formula ψ, one can construct an NBW $\mathcal{A}_\psi$ that accepts exactly all the computations that satisfy ψ. The size of $\mathcal{A}_\psi$ is, in the worst case, exponential in $|\psi|$ [37].

We model systems by *Kripke structures*. A Kripke structure is a tuple $K = \langle AP, W, W_0, R, l\rangle$, where W is the set of states, $R \subseteq W \times W$ is a total transition relation (that is, for every $w \in W$, there is at least one state w' such that $R(w, w')$), $W_0 \subseteq W$ is a set of initial states, and $l : W \to 2^{AP}$ maps each state to the set of atomic propositions that hold in it. A *path* in K is a (finite or infinite) sequence $w_0, w_1, \ldots$ of states in W such that $w_0 \in W_0$ and for all $i \geq 0$ we have $R(w_i, w_{i+1})$. A *computation* of K is a (finite or infinite) sequence $l(w_0), l(w_1), \ldots$ of assignments in 2^{AP} for a path $w_0, w_1, \ldots$ in K. We assume that different states of K are labeled differently. That is, for all states $w, w' \in W$ such that $w \neq w'$, we have $l(w) \neq l(w')$. The assumption makes our setting cleaner, as it amount to working with deterministic systems, so all the nondeterminism and probabilistic choices are linked to the specification and the distribution of the inputs, which is our focus. The simplest way to adjust nondeterministic systems to our setting is to add atomic propositions that resolve nondeterminism. The language of K, denoted $L(K)$, is the set of its infinite computations. We say that K satisfies an LTL formula ψ, denoted $K \models \psi$, if all the computations of K satisfy ψ, thus $L(K) \subseteq L(\psi)$. We define the size of a Kripke structure K, denoted $|K|$, as $|W| + |R|$.

For a set AP of atomic propositions, we define the Kripke structure $K_{AP} = \langle AP, 2^{AP}, 2^{AP}, 2^{AP} \times 2^{AP}, l\rangle$, where $l(w) = w$ for all $w \in 2^{AP}$. Thus, K_{AP} is a 2^{AP}-clique satisfying $L(K_{AP}) = (2^{AP})^\omega$.

2.2 Safety, Liveness, and Counterable Languages

Consider an alphabet Σ, a language $L \subseteq \Sigma^\omega$, and a finite word $u \in \Sigma^*$. We say that u is a *prefix for* L if it can be extended to an infinite word in L, thus there is $v \in \Sigma^\omega$ such that $uv \in L$. Then, u is a *bad-prefix for* L if it cannot be extended to an infinite word in L, thus for every $v \in \Sigma^\omega$, we have that $uv \notin L$. Note that if u is a bad-prefix, so are all its finite extensions. We denote by *pref*(L) the set of all prefixes for L.

The following classes of languages have been extensively studied (c.f., [2, 3]). A language $L \subseteq \Sigma^\omega$ is a *safety* language if every word not in L has a bad-prefix. For example, $\{a^\omega\}$ over $\Sigma = \{a, b, c\}$ is safety, as every word not in L has a bad-prefix – one that contains the letter b or c. A language L is a *liveness* language if every finite word can be extended to a word in L. Thus, *pref*$(L) = \Sigma^*$. For example, the language $(a + b + c)^* \cdot a^\omega$ is a liveness language: by concatenating a^ω to every word in Σ^*, we end up with a word in the language. When L is not liveness, namely *pref*$(L) \neq \Sigma^*$, we say that L is *counterable*. Note that while a liveness language has no bad-prefix, a counterable language has at least one bad-prefix. For example, $L = a^* \cdot b \cdot (a + b + c)^\omega$ is a counterable language. Indeed, c is a bad-prefix for L. It is not hard to see that if L is safety and $L \neq \Sigma^\omega$, then L is counterable. The other direction does not hold. For example, L above is not safety, as the word a^ω has no bad-prefix.

We extend the definitions and classes above to specifications given by LTL formulas or by NBWs. For example, an LTL formula ψ is counterable iff $L(\psi)$ is counterable.

3 Probabilistic and Relative Counterability

In this section we introduce and make some observations on two variants to counterability. The first variant adds a probabilistic component to the definitions. The second makes them relative to a Kripke structure. We also consider the combination of the probabilistic and relative variants.

3.1 Probabilistic Counterability

For a finite or countable set X, a *probability distribution* on X is a function $Pr : X \to [0,1]$ assigning a probability to each element in X. Accordingly, $\sum_{x \in X} Pr(x) = 1$. A *finite Markov chain* is a tuple $\mathcal{M} = \langle V, p_{in}, p \rangle$, where V is a finite set of states, $p_{in} : V \to [0,1]$ is a probabilistic distribution on V that describes the probability of a path to start in the state, and $p : V \times V \to [0,1]$ is a function describing a distribution over the transitions. Formally, for $v \in V$, let $p_v : V \to [0,1]$ be such that $p_v(u) = p(v,u)$ for all $u \in V$. For all $v \in V$, the function p_v is a probabilistic distribution on V. The Markov chain $\mathcal{M}$ induces the directed graph $G = \langle V, E \rangle$ in which for all $u, v \in V$, we have that $(u,v) \in E$ iff $p(u,v) > 0$. Thus, G includes transitions that have a positive probability in $\mathcal{M}$. When we talk about the SCCs of $\mathcal{M}$, we refer to these of G.

A *random walk* on $\mathcal{M}$ is an infinite path $v_0, v_1, \ldots$ in G such that v_0 is drawn at random according to p_{in} and the i-th state v_i is drawn at random according to $p_{v_{i-1}}$. More formally, there is a probability space $\langle V^\omega, \mathcal{F}, Pr_{\mathcal{M}} \rangle$ defined on the set V^ω of infinite sequences of states. The family of measurable sets $\mathcal{F}$ is the σ-algebra (also called *Borel field*) generated by $C = \{C(x) : x \in V^*\}$ where $C(x)$ is the set of infinite sequences with prefix x. The measure $Pr_{\mathcal{M}}$ is defined on C (and can be extended uniquely to the rest of $\mathcal{F}$) as follows: $Pr_{\mathcal{M}}[C(x_0, \ldots, x_n)] = p_{in}(x_0) \cdot p(x_0, x_1) \cdot \ldots \cdot p(x_{n-1}, x_n)$. For more background on the construction of this probability space, see, for example, [20]. A random walk on $\mathcal{M}$ from a state $v \in V$ is a random walk on the Markov chain $\mathcal{M}^v = \langle V, p^v_{in}, p \rangle$, where $p^v_{in}(v) = 1$.

▶ **Lemma 1** ([20]). *Consider a Markov chain $M = \langle V, p_{in}, p \rangle$ and a state $v \in V$.*

1. *An infinite random walk on $\mathcal{M}^v$ reaches some ergodic SCC with probability* 1.
2. *Once a random walk on $\mathcal{M}^v$ reaches an ergodic SCC, it visits all its states infinitely often with probability* 1.

A *labeled finite Markov chain* is a tuple $\mathcal{S} = \langle \Sigma, V, p_{in}, p, \tau \rangle$, where Σ is an alphabet, $\mathcal{M} = \langle V, p_{in}, p \rangle$ is a finite Markov chain, and $\tau : V \to \Sigma$ maps each state in V to a letter in Σ. We extend τ to paths in the expected way, thus for $\pi = v_0, v_1, \ldots \in V^\omega$, we define $\tau(\pi) = \tau(v_0), \tau(v_1), \ldots$. A random walk on $\mathcal{S}$ is a random walk on $\mathcal{M}$. The chain $\mathcal{S}$ induces a probability space on Σ^ω, induced from $\mathcal{M}$. That is, for $L \subseteq \Sigma^\omega$, we have $Pr_{\mathcal{S}}[L] = Pr_{\mathcal{M}}[\{\pi \in V^\omega : \tau(\pi) \in L\}]$. It is known that ω-regular languages are measurable in $\mathcal{S}$ (c.f., [36]).

Consider an alphabet Σ. A random word over Σ is a word in which for all indices i, the i-th letter is drawn from Σ uniformly at random. We denote by $Pr[L]$ the probability of a language $L \subseteq \Sigma^\omega$ in this uniform distribution. For a finite word $u \in \Sigma^*$, we denote by $Pr^u[L]$ the probability that a word obtained by concatenating an infinite random word to u is in L. Formally, $Pr^u[L] = Pr[\{v \in \Sigma^\omega : uv \in L\}]$. For example, let $\Sigma = \{a, b, c\}$ and $L = a^* \cdot b \cdot (a + b + c)^\omega$. Then, $Pr[L] = \sum_{i=1}^{\infty} \frac{1}{3^i} = \frac{1}{2}$, $Pr^a[L] = Pr[L] = \frac{1}{2}$, $Pr^{ab}[L] = 1$, and $Pr^c[L] = 0$.

Consider a language $L \subseteq \Sigma^\omega$. We say that a finite word $u \in \Sigma^*$ is a *prob-bad-prefix* for L if $Pr^u[L] = 0$. That is, u is a prob-bad-prefix if an infinite word obtained by continuing u randomly is almost surely not in L. We say that L is *prob-counterable* if it has a prob-bad-prefix. Consider for example the language $L = a \cdot (a+b)^\omega + b^\omega$ over $\Sigma = \{a, b\}$. All the words $u \in b^+$ are not bad-prefixes for L, but are prob-bad-prefixes, as $Pr^u[L] = Pr[b^\omega] = 0$. As another example, consider the LTL formula $\psi = (req \wedge GF grant) \vee (\neg req \wedge FG \neg grant)$. The formula ψ is a liveness formula and does not have a bad-prefix. Thus, ψ is not counterable. All finite computations in which a request is not sent in the beginning of the computation are, however, prob-bad-prefixes for ψ, as the probability of satisfying $FG \neg grant$ is 0. Hence, ψ is prob-counterable.

Prob-counterability talks about prefixes after which the probability of being in L is 0. We can relate such prefixes to words that lead to rejecting ergodic SCCs in DPWs that recognize L, giving rise to the following alternative definitions:

▶ **Theorem 2.** *Consider an ω-regular language L. The following are equivalent:*
1. *The language L is prob-counterable.*
2. *$Pr[L] < 1$. That is, the probability of an infinite random word to be in L is strictly smaller than 1.*
3. *Every DPW that recognizes L has a rejecting ergodic SCC.*

Analyzing the SCCs of DPWs for L also implies that ω-regular languages have a "safety-like" behavior in the following probabilistic sense:

▶ **Theorem 3.** *For every ω-regular language L, we have $Pr[\{v \in \Sigma^\omega : v \notin L$ and v does not have a prob-bad-prefix$\}] = 0$.*

Note that if L is prob-counterable, thus $Pr[v \notin L] > 0$, then $Pr[v$ has a prob-bad-prefix $\mid v \notin L] = 1$.

3.2 Relative Counterability

Recall that the standard definitions of bad-prefixes consider extensions in Σ^ω. When $\Sigma = 2^{AP}$ and the language L is examined with respect to a Kripke structure K over AP, it is interesting to restrict attention to extensions that are feasible in K. Consider a finite word $u \in \Sigma^*$. We say that u is a *bad-prefix for L with respect to K* (K*-bad-prefix*, for short) if u is a finite computation of K that cannot be extended to a computation of K that is in L. Thus, $u \in pref(L(K)) \setminus pref(L(K) \cap L)$. We say that L is *safety with respect to K* (K*-safety*, for short) if every computation of K that is not in L has a K-bad-prefix. We say that L is *counterable with respect to K* (K*-counterable*, for short) if the language L has a K-bad-prefix. Thus, $pref(L(K)) \not\subseteq pref(L(K) \cap L)$.

▶ **Theorem 4.** *Consider an ω-regular language $L \subseteq (2^{AP})^\omega$.*
1. *L is safety iff L is K-safety for every Kripke structure K over AP.*
2. *For every Kripke structure K over AP, we have that L is K-safety iff $L(K) \cap L$ is safety.*

Recall that if $L \subseteq \Sigma^\omega$ is safety and $L \neq \Sigma^\omega$, then L is counterable. Also, if L is K-safety and $L(K) \not\subseteq L$ then L is K-counterable. Note that it is possible that $L(K) \cap L$ is counterable but L is not K-counterable. For example, we can choose K and L such that $L(K) \subseteq L \neq (2^{AP})^\omega$. Then, L is not K-counterable, but a word u that is not a computation of K is a bad-prefix for $L(K)$, making it also a bad-prefix for $L(K) \cap L$. Hence, $L(K) \cap L$ is counterable.

3.3 Probabilistic Relative Counterability

We combine the probabilistic and relative definitions. Consider a Kripke structure $K = \langle AP, W, W_0, R, l\rangle$. A *$K$-walk-distribution* is a tuple $P = \langle p_{in}, p\rangle$ such that $M_{K,P} = \langle 2^{AP}, W, p_{in}, p, l\rangle$ is a labeled Markov chain that induces a graph that agrees with K. Thus, $p_{in}(w) > 0$ iff $w \in W_0$, and $p(w, w') > 0$ iff $R(w, w')$. A random walk on K with respect to P is a random walk on the Markov chain $\langle W, p_{in}, p\rangle$. We define the probability of an ω-regular language $L \subseteq (2^{AP})^\omega$ with respect to K and P as $Pr_{K,P}[L] = Pr_{M_{K,P}}[L]$. Namely, $Pr_{K,P}[L]$ is the probability that a computation obtained by a random walk on K with respect to P is in L. Let u be a finite computation of K and let $w_0, \ldots, w_k \in W^*$ be such that

$u = l(w_0), \ldots, l(w_k)$. We say that an infinite computation u' is a continuation of u with a random walk on K with respect to P if $u' = l(w_0), \ldots, l(w_{k-1}), l(w'_0), l(w'_1), \ldots$, where $w'_0, w'_1, \ldots$ is obtained by a random walk on K from w_k with respect to P (recall that we assume that different states in K are labeled differently). We define $Pr^u_{K,P}[L]$ as the probability that a computation obtained by continuing u with a random walk on K with respect to P is in L. Formally, we define $Pr^u_{K,P}$ using a conditional probability: $Pr^u_{K,P}[L] = Pr_{K,P}[L|$ the word starts with $u]$. We extend the definition to every labeled Markov chain $\mathcal{M}$; that is $Pr^u_{\mathcal{M}}[L] = Pr_{\mathcal{M}}[L|$ the word starts with $u]$. Thus, $Pr^u_{K,P}[L] = Pr^u_{M_{K,P}}[L]$.

Consider a Kripke structure K over AP and an ω-regular language $L \subseteq (2^{AP})^\omega$. We say that $u \in (2^{AP})^*$ is a *prob-bad-prefix for L with respect to K* (*K-prob-bad-prefix*, for short) if u is a finite computation of K such that $Pr^u_{K,P}[L] = 0$, for some K-walk-distribution P. Thus, a computation obtained by continuing u with some random walk on K is almost surely not in L. As we show in Lemma 5 below, the existential quantification on the K-walk-distribution P can be replaced by a universal one, or by the specific K-walk-distribution that traverses K uniformly at random. We say that L is *prob-counterable with respect to K* (*K-prob-counterable*, for short) if it has a K-prob-bad-prefix.

Note that if L is counterable, then L is prob-counterable. Also, if L is K-counterable then L is K-prob-counterable. As we have seen above, a language may be prob-counterable but not counterable. Taking $K = K_{AP}$, this implies that a language may be K-prob-counterable but not K-counterable. As an example with an explicit dependency in K, consider the counterable LTL formula $\psi = G(req \rightarrow X\, ack) \wedge FG(idle)$. Let K over $AP = \{req, ack, idle\}$ be such that the atomic propositions in AP are mutually exclusive and $L(K)$ contains exactly the computations that start in *req* and in which every *req* is immediately followed by *ack*, or computations in which *idle* is always valid. Note that while ψ is not K-counterable, it is K-prob-counterable, as every finite computation of K that starts with *req* is a K-prob-bad-prefix for ψ.

Consider an NBW $\mathcal{A}$. By [36, 11], deciding whether $Pr_{K,P}[L(\mathcal{A})] = 0$ or whether $Pr_{K,P}[L(\mathcal{A})] = 1$ is independent of the K-walk-distribution P. Consequently, we have the following.

▶ **Lemma 5.** [36, 11] *Let u be a finite computation of a Kripke structure K over AP, and let $L \subseteq (2^{AP})^\omega$ be an ω-regular language. For all pairs P and P' of K-walk-distributions, we have that $Pr^u_{K,P}[L] = 0$ iff $Pr^u_{K,P'}[L] = 0$ and $Pr^u_{K,P}[L] = 1$ iff $Pr^u_{K,P'}[L] = 1$.*

We can now point to equivalent definitions of K-prob-counterability.

▶ **Theorem 6.** *Consider an ω-regular language $L \subseteq (2^{AP})^\omega$ and a Kripke structure K over AP. The following are equivalent:*

1. *The language L is K-prob-counterable.*
2. *There is a finite computation u of K and a K-walk-distribution P such that $Pr^u_{K,P}[L] < 1$.*
3. *There is a finite computation u of K such that for all K-walk-distribution P, we have $Pr^u_{K,P}[L] < 1$.*
4. *There is a K-walk-distribution P s.t. $Pr_{K,P}[L] < 1$.*
5. *For all K-walk-distributions P, we have $Pr_{K,P}[L] < 1$.*

We can also generalize Theorem 3, and show that ω-regular languages have "safety-like" behaviors also with respect to Kripke structures, in the following probabilistic sense:

▶ **Theorem 7.** *Consider an ω-regular language $L \subseteq (2^{AP})^\omega$, a Kripke structure K over AP, and a K-walk-distribution P. Then, $Pr_{K,P}[\{u \in (2^{AP})^\omega : u \notin L$ and u does not have a K-prob-bad-prefix for $L\}] = 0$.*

If L is also K-prob-counterable then we also have $Pr_{K,P}[u \text{ has a } K\text{-prob-bad-prefix for } L \mid u \notin L] = 1$ for every K-walk-distribution P.

Conceptually, Theorem 6 implies that if an error has a positive probability to occur in a random execution of the system, then the specification is prob-counterable with respect to the system. Theorem 7 then suggests that in this case, a computation of the system that does not satisfy the specification, almost surely has a prob-bad-prefix with respect to the system. Thus, almost all the computations that violate the specification start with a prob-bad-prefix with respect to the system. Hence, attempts to find and return to the user such bad-prefixes are very likely to succeed.

4 Deciding Liveness

Recall that a language L is counterable iff L is not liveness. As discussed in Section 1, the complexity of the problem of deciding whether a given LTL formula is liveness is open [4]. In this section we solve this problem and prove that it is EXPSPACE-complete. The result would be handy also for our study of the probabilistic and relative variants.

▶ **Theorem 8.** *The problem of deciding whether a given LTL formula is liveness is EXPSPACE-complete.*

Proof. The upper-bound is known [35], and follows from the fact that every LTL formula ψ can be translated to an NBW $\mathcal{A}_\psi$ with an exponential blow-up [37]. By removing empty states from $\mathcal{A}_\psi$ and making all other states accepting, we get an NFW for $\mathit{pref}(L(\psi))$, which is universal iff ψ is liveness. The latter can be checked on-the-fly and in PSPACE, implying the EXPSPACE upper bound.

For the lower bound, we show a reduction from an exponent version of the *tiling problem*, defined as follows. We are given a finite set T, two relations $V \subseteq T \times T$ and $H \subseteq T \times T$ of tiles, an initial tile t_0, a final tile t_f, and a bound $n > 0$. We have to decide whether there is some $m > 0$ and a tiling of a $2^n \times m$-grid such that (1) The tile t_0 is in the bottom left corner and the tile t_f is in the top left corner, (2) A horizontal condition: every pair of horizontal neighbors is in H, and (3) A vertical condition: every pair of vertical neighbors is in V. Formally, we have to decide whether there is a function $f : \{0, \ldots, 2^n - 1\} \times \{0, \ldots, m-1\} \to T$ such that (1) $f(0,0) = t_0$ and $f(0, m-1) = t_f$, (2) For every $0 \leq i \leq 2^n - 2$ and $0 \leq j \leq m-1$, we have that $(f(i,j), f(i+1,j)) \in H$, and (3) For every $0 \leq i \leq 2^n - 1$ and $0 \leq j \leq m-2$, we have that $(f(i,j), f(i,j+1)) \in V$. When n is given in unary, the problem is known to be EXPSPACE-complete.

We reduce this problem to the problem of deciding whether an LTL formula is not liveness. Given a tiling problem $\tau = \langle T, H, V, t_0, t_f, n \rangle$, we construct a formula φ such that τ admits tiling iff φ has a *good-prefix* – one all whose extensions satisfy φ. Formally, $x \in \Sigma^*$ is a good prefix for φ iff for all $y \in \Sigma^\omega$, we have that $x \cdot y$ satisfies φ. Therefore, for $\psi = \neg\varphi$, we have τ admits tiling iff ψ is not liveness. The idea is to encode a tiling as a word over T, consisting of a sequence of rows (each row is of length 2^n). Such a word represents a proper tiling if it starts with t_0, has a last row that starts with t_f, every pair of adjacent tiles in a row are in H, and every pair of tiles that are 2^n tiles apart are in V. The difficulty is in relating tiles that are far apart. To do that we represent every tile by a block of length n, which encodes the tile's position in the row. Even with such an encoding, we have to specify a property of the form "for every i, if we meet a block with position counter i, then the next time we meet a block with position counter i, the tiles in the two blocks are in V". Such a property can be expressed in an LTL formula of polynomial length, but there are exponentially many

i's to check. The way to use liveness in order to mimic the universal quantification on all i's is essentially the following: the good prefix for φ encodes the tiling. The set of atomic propositions in φ includes a proposition \$ that is not restricted beyond this prefix. The property that checks V then has to hold in blocks whose counter equals the counter of the block that starts at the last \$ in the computation. Thus, universal quantification in i is replaced by the explicit universal quantification on suffixes in the definition of good prefixes. A detailed description of the reduction is included in the full version of the paper. ◀

When a language is given by an NBW, the complexity of deciding its liveness is much simpler:

▶ **Theorem 9.** *The problem of deciding whether a given NBW is liveness is PSPACE-complete.*

5 On Counterability

In this section we study the problem of deciding whether a given language is counterable as well as the length of short bad-prefixes and their detection. In order to complete the picture, we also compare the results to those of safety languages.

We start with the complexity of deciding safety and counterability. The results for safety are from [35]. These for counterability follow from Theorems 8 and 9 and the fact that L is counterable iff it is not liveness.

▶ **Theorem 10.** *Consider a language L.*

1. [35] *The problem of deciding whether L is safety is PSPACE-complete for L given by an LTL formula or by an NBW.*
2. *The problem of deciding whether L is counterable is EXPSPACE-complete for L given by an LTL formula and is PSPACE-complete for L given by an NBW.*

We find Theorem 10 surprising: both safety and counterability ask about the existence of bad-prefixes. In safety, a bad-prefix should exist to all bad words. In counterability, not all bad words have a bad-prefix, but at least some should have. Theorem 10 implies that there is something in LTL, yet not in NBWs, that makes the second type of existence condition much more complex.

We now turn to study the length of shortest bad-prefixes. Both (non-valid) safety and counterable languages have bad-prefixes. As we show, however, the complexity of counterability continues, and a tight bound on shortest bad-prefixes for counterable languages is exponentially bigger than that of safety languages. The gap follows from our ability to construct a *fine automaton* $\mathcal{A}_\psi^{fine}$ for all safety LTL formulas ψ [21]. The NFW $\mathcal{A}_\psi^{fine}$ is exponential in $|\psi|$, it accepts only bad-prefixes for ψ, and each computation that does not satisfy ψ has at least one bad-prefix accepted by $\mathcal{A}_\psi^{fine}$. A shortest witness to the nonemptiness of $\mathcal{A}_\psi^{fine}$ can serve as a bad-prefix. On the other hand, nothing is guaranteed about the behavior of $\mathcal{A}_\psi^{fine}$ when constructed for a non-safety formula ψ, thus it is of no help in the case of counterable languages that are not safety. In particular, the LTL formula used in the proof of Theorem 8 is neither safety nor its complement is safety, thus its doubly-exponential shortest bad-prefix does not contradict upper bounds known for these classes of languages.

▶ **Theorem 11.** *The length of shortest bad-prefixes for a language given by an LTL formula ψ is tightly exponential in $|\psi|$ in case ψ is safety, and is tightly doubly-exponential in $|\psi|$ in case ψ is counterable.*

When the specification formalism is automata, the difference between safety and counterable languages disappears:

▶ **Theorem 12.** *The length of shortest bad-prefixes for a safety or a counterable language given by an NBW $\mathcal{A}$ is tightly exponential in $|\mathcal{A}|$.*

6 On Relative Counterability

In this section we add a Kripke structure K to the setting and study K-counterablity and shortest K-bad-prefixes. Our results use variants of the product of automata for K and L, and we first define this product below. Consider a Kripke structure $K = \langle AP, W, W_0, R, l\rangle$ and an NBW $\mathcal{A} = \langle 2^{AP}, Q, Q_0, \delta, \alpha\rangle$. Essentially, the states of the product $\mathcal{A}_{K\times\mathcal{A}}$ are pairs in $W \times Q$. Recall that the states in K are differently labeled. Thus, we can define the product so that whenever it reads a letter in 2^{AP}, its next W-component is determined. Formally, we define the NBW $\mathcal{A}_{K\times\mathcal{A}} = \langle 2^{AP}, (W \cup \{s_0\}) \times Q, \{s_0\} \times Q_0, \rho, W \times \alpha\rangle$, where for all $\sigma \in 2^{AP}$, we have $\langle w', q'\rangle \in \rho(\langle s_0, q\rangle, \sigma)$ iff $w' \in W_0$ and $l(w') = \sigma$ and $q' \in \delta(q, \sigma)$, and for $w \in W$ we have $\langle w', q'\rangle \in \rho(\langle w, q\rangle, \sigma)$ iff $R(w, w')$ and $l(w') = \sigma$ and $q' \in \delta(q, \sigma)$. Thus, when the product $\mathcal{A}_{K\times\mathcal{A}}$ proceeds from state $\langle w, q\rangle$ with σ, its new W-component is the single successor of w that is labeled σ, paired with the σ-successors of q. It is easy to see that $L(\mathcal{A}_{K\times\mathcal{A}}) = L(K) \cap L(\mathcal{A})$. When $\mathcal{A}$ corresponds to an LTL formula ψ (that is, $L(\mathcal{A}) = L(\psi)$), we denote the product by $\mathcal{A}_{K\times\psi}$.

We start with the problem of deciding relative safety. By Theorem 4, a language L is K-safety iff $L(K) \cap L$ is safety. Thus, the check can be reduced to checking the safety of $\mathcal{A}_{K\times\mathcal{A}}$ (respectively $\mathcal{A}_{K\times\psi}$). This check, however, if done naively, is PSPACE in $|\mathcal{A}_{K\times\mathcal{A}}|$ (respectively $|\mathcal{A}_{K\times\psi}|$), which is PSPACE in $|K|$. The technical challenge is to find a more efficient way to do the check, and the one we describe in the proof is based on decomposing $\mathcal{A}_{K\times\mathcal{A}}$ so that the complementation that its safety check involves is circumvented. As for the lower bound, note that using the Kripke structure K_{AP}, one can reduce traditional safety to relative safety.[3] Our reduction, however, shows that the complexity of deciding K-safety coincides with that of model checking in both its parameters.

▶ **Theorem 13.** *Consider a Kripke structure K over AP and a language $L \subseteq (2^{AP})^\omega$. The problem of deciding whether L is K-safety is PSPACE-complete for L given by an NBW or by an LTL formula. In both cases it can be done in time linear and space polylogarithmic in $|K|$.*

We continue to relative counterability. We first show that the complexity of deciding counterability is carried over to the relative setting. For the upper bound, note that a language L is K-counterable iff $\mathit{pref}(L(K)) \cap \mathit{comp}(\mathit{pref}(L(K) \cap L)) \neq \emptyset$. Again, this check, if done naively, is PSPACE in $|K|$, and the challenge in the proof is to use the deterministic behavior of $\mathcal{A}_{K\times\mathcal{A}}$ with respect to the W-component of its states in order to avoid a blow-up in K in its complementation.

▶ **Theorem 14.** *Consider a Kripke structure K over AP and a language $L \subseteq (2^{AP})^\omega$. The problem of deciding whether L is K-counterable and finding a shortest K-bad-prefix is PSPACE-complete for L given by an NBW and is EXPSPACE-complete for L given by an LTL formula. In both cases it can be done in time linear and space polylogarithmic in $|K|$.*

[3] Indeed K_{AP} is exponential in $|AP|$, but safety is known to be PSPACE-hard also when the number of atomic propositions is fixed.

We now study the length of K-bad-prefixes. The upper bounds follow from the proof of Theorem 14, and for the lower ones, we rely on K_{AP} and the constructions described in the proofs of Theorems 11 and 12 in order to prove the dependency on $|\psi|$ and $|\mathcal{A}|$, and describe a family of Kripke structures requiring linear dependency in $|K|$:

▶ **Theorem 15.** *The length of a shortest K-bad-prefix for a K-counterable language L is tightly doubly-exponential in $|\psi|$, in case L is given by means of an LTL formula ψ, and is tightly exponential in $|\mathcal{A}|$, in case L is given by an NBW $\mathcal{A}$. In both cases, it is also tightly linear in $|K|$.*

Interestingly, when the LTL formula is K-safety, deciding its K-counterability and finding a K-bad-prefix can be done more efficiently, and its K-bad-prefixes are shorter. For the decidability problem, note that if an LTL formula ψ is K-safety, then ψ is K-counterable iff $K \not\models \psi$. Also, by Theorem 4, the fact ψ is K-safety implies that the NBW $\mathcal{A}_{K\times\psi}$ is safety, which is useful in the construction of fine automata. Formally, we have the following.

▶ **Theorem 16.** *Let K be a Kripke structure and let ψ be a K-safety LTL formula. Deciding whether ψ is K-counterable and finding a K-bad-prefix is PSPACE-complete in $|\psi|$. The length of shortest K-bad-prefixes is tightly exponential in $|\psi|$.*

Note that the space complexity of the algorithm described in the proof of Theorem 16 in also polylogarithmic in $|K|$. Finally, when the specification formalism is automata, the difference between K-safety and K-counterable languages disappears, and a short K-bad-prefix for a K-safety or K-counterable language given by an NBW $\mathcal{A}$ is tightly exponential in $|\mathcal{A}|$ and linear in $|K|$.

7 On Probabilistic Relative Counterability

In this section we study K-prob-counterability. By Theorem 6, an ω-regular language L is K-prob-counterable iff $Pr_{K,P}[L] < 1$ for some K-walk-distribution P. This, together with [11], imply the upper bound for the corresponding decision problem:

▶ **Theorem 17.** *Consider a language $L \subseteq (2^{AP})^\omega$ and a Kripke structure K over AP. Deciding whether L is K-prob-counterable can be done in time $O(|K| \cdot 2^{O(|L|)})$ or in space polynomial in $|L|$ and polylogarithmic in $|K|$, for L given by an LTL formula ψ, in which case $|L| = |\psi|$, or by an NBW $\mathcal{A}$, in which case $|L| = |\mathcal{A}|$. In both cases, the problem is PSPACE-complete.*

Thus, deciding whether an LTL formula is K-prob-counterable is exponentially easier than in the non-probabilistic case.

Recall that the study of K-counterability involved reasoning about the product of K and a nondeterministic automaton for the language. In the probabilistic setting, we need the automaton for the language to be deterministic. Let $\mathcal{D} = \langle 2^{AP}, S, s_0, \delta_{\mathcal{D}}, \alpha\rangle$ be a DPW for a language L and let $K = \langle AP, W, W_0, R, l\rangle$ be a Kripke structure. We define $\mathcal{D}_{K\times\mathcal{D}} = \langle 2^{AP}, W \times S, W_0 \times \{s_0\}, \rho, \alpha'\rangle$ as the product DPW of K and $\mathcal{D}$. Formally, we have $\alpha'(\langle w, s\rangle) = \alpha(s)$, and $\langle w', s'\rangle \in \rho(\langle w, s\rangle, \sigma)$ iff $[l(w) = \sigma$, $R(w, w')$ and $s' = \delta_{\mathcal{D}}(s, \sigma)]$. Note that $L(\mathcal{D}_{K\times\mathcal{D}}) = L(K) \cap L(\mathcal{D})$. Also, note that all of the successors of a state in $\mathcal{D}_{K\times\mathcal{D}}$ share the same second component.

In the probabilistic setting, we also need to define the product as a Markov chain. Let $\mathcal{D}$ and K be as above and let $P = \langle p_{in}, p\rangle$ be a K-walk-distribution. We define a labeled Markov chain $M' = \langle 2^{AP}, W \times S, p'_{in}, p', l'\rangle$, where $p' : (W \times S) \times (W \times S) \to [0, 1]$ is such that

$p'(\langle w,s\rangle,\langle w',s'\rangle) = p(\langle w,w'\rangle)$ if $\delta_{\mathcal{D}}(s,l(w)) = s'$, and otherwise $p'(\langle w,s\rangle,\langle w',s'\rangle) = 0$. Also, $p'_{in} : W \times S \to [0,1]$ is such that $p'_{in}(\langle w,s\rangle) = p_{in}(w)$ if $s = s_0$, and otherwise $p'_{in}(\langle w,s\rangle) = 0$. Finally, $l' : W \times S \to 2^{AP}$ is such that $l'(\langle w,s\rangle) = l(w)$. Thus, p'_{in} and p' attribute the states of $M = M_{K,P}$ by the deterministic behavior of $\mathcal{D}$. In particular, note that for every $\langle w,s\rangle \in W \times S$, we have $\sum_{\langle w',s'\rangle \in W\times S} p'(\langle w,s\rangle,\langle w',s'\rangle) = \sum_{w'\in W} p\langle w,w'\rangle = 1$.

Let $g : W \times S \to W$ be a function that projects pairs in $W \times S$ on their W-component, namely $g(\langle w,s\rangle) = w$. Consider random walks $X = X_1, X_2 \ldots$ and $X' = X'_1, X'_2 \ldots$ on M and M', respectively. Let $Y = Y_1, Y_2, \ldots$ be the projection of X' on W, thus $Y_i = g(X'_i)$ for $i = 1, 2, \ldots$. Note that the processes X and Y both take values in W^ω and that they have the same distribution. Therefore, we have $Pr_M[L] = Pr_{M'}[L]$. Also, for a finite computation $u = l(w_0), \ldots, l(w_n)$ of K, we have $Pr^u_M[L] = Pr^u_{M'}[L]$. Note that $w_0, \ldots, w_n$ induces a single finite path $\langle w_0,s_0\rangle, \ldots, \langle w_n,s_n\rangle$ in $\mathcal{D}_{K\times\mathcal{D}}$, and that every infinite path in K induces a single infinite path in $\mathcal{D}_{K\times\mathcal{D}}$. For a finite computation u, let $reach(u)$ be the state reached in $\mathcal{D}_{K\times\mathcal{D}}$ after traversing u. Thus, $reach(u) = \langle w_n,s_n\rangle$. By the above, $Pr^u_{M'}[L]$ is the probability that a random walk in M' from $reach(u)$ is an accepting run in $\mathcal{D}_{K\times\mathcal{D}}$. For a state x of M', we denote by γ_x the probability that a random walk from x in M' is an accepting run in $\mathcal{D}_{K\times\mathcal{D}}$. Note that a finite computation u is a K-prob-bad-prefix iff $\gamma_{reach(u)} = 0$.

▶ **Lemma 18.** *Deciding whether γ_x is 0, 1 or in $(0,1)$, for all states x in M', can be done in time linear in $|\mathcal{D}|$ and in $|K|$ or in space polylogarithmic in $|\mathcal{D}|$ and in $|K|$. Furthermore, the probability γ_x can be calculated in time polynomial in $|\mathcal{D}|$ and in $|K|$.*

We turn to study the complexity of probabilistic relative counterability of ω-regular languages. Handling a language given by an NBW can proceed by an exponential translation to a DPW [31]. For languages given by LTL formulas, going to DPWs involves a doubly-exponential blow-up. We show that in order to find a K-prob-bad-prefix for an LTL formula, we can carefully proceed according to the syntax of the formula and do exponentially better than an algorithm that translates the formulas to automata. We note that the PSPACE-hardness in Theorem 17 is by a reduction from the universality problem. Thus, we cannot hope to obtain a PSPACE algorithm by translating LTL formulas to NBWs, unless the structure of the latter is analyzed to a level in which it essentially follows the structure of the LTL formula (see, for example, [12] for such an approach applied in probabilistic LTL model checking).

7.1 On probabilistic relative counterability of NBWs

We start with an algorithm for finding a shortest K-prob-bad-prefix for a language given by an NBW $\mathcal{A}$. For that, we need to find a shortest word whose path in M' reaches a state x for which $\gamma_x = 0$. By Lemma 18, we thus have the following:

▶ **Theorem 19.** *Consider an NBW $\mathcal{A}$ over the alphabet 2^{AP} and a Kripke structure K over AP. Finding a shortest K-prob-bad-prefix for $L(\mathcal{A})$ can be done in space polynomial in $|\mathcal{A}|$ and polylogarithmic in $|K|$ or in time exponential in $|\mathcal{A}|$ and linear in $|K|$. Furthermore, the length of the shortest K-prob-bad-prefix for $L(\mathcal{A})$ is tightly exponential in $|\mathcal{A}|$ and tightly linear in $|K|$.*

Consider a Kripke structure K over AP and a language $L \subseteq (2^{AP})^\omega$ that is K-prob-counterable. In practice, the user of the model-checking tool often has some estimation of the likelihood of every transition in the system. That is, we assume that the user knows what the typical K-walk-distribution P in a typical behavior of the system is. Clearly, there is a trade-off between the length of a counterexample and its "precision", in the sense that the

longer a finite prefix of an erroneous computation is, the larger is the probability in which it is a K-prob-bad-prefix. We want to allow the user to play with this trade-off and thus define the following two problems:

- The *shortest bounded-prob-K-bad-prefix* problem is to return, given K, L, and $0 < \gamma < 1$, a shortest finite computation u of K such that $Pr^u_{K,P}[L] < \gamma$.
- The *bounded-length prob-K-bad-prefix* problem is to return, given K, L, and $m \geq 1$, a finite computation u of K such that $|u| \leq m$ and $Pr^u_{K,P}[L]$ is minimal.

Using Lemma 18, we can carefully reduce both problems to classical problems in graph algorithms, applied to $\mathcal{D}_{K \times \mathcal{D}}$:

▶ **Theorem 20.** *The shortest bounded-prob-K-bad-prefix and the bounded-length prob-K-bad-prefix problems, for a language given by an NBW $\mathcal{A}$, can be solved in time exponential in $|\mathcal{A}|$ and polynomial in $|K|$.*

We note that using our construction of M', together with Lemma 18, we can reduce the calculation of $Pr_{K,P}[L(\mathcal{A})]$ or the problem of its classification to 1, 0, or $(0, 1)$ to a sequence of calculations in M', simplifying the known result of [11] (Theorem 4.1.7 there).

7.2 On Probabilistic Relative Counterability of LTL formulas

We describe an algorithm for finding a K-prob-bad-prefix for an LTL formula θ. Like the model-checking algorithm in [11], our algorithm proceeds by iteratively replacing the innermost temporal subformula of θ by a fresh atomic proposition and adjusting K so that the probability of a computation obtained by a random walk to satisfy the specification is maintained. In more detail, we construct a sequence $K^0, K^1, \ldots$ of Kripke structures and a sequence $\theta^0, \theta^1, \ldots$ of LTL formulas such that $K^0 = K$ and $\theta^0 = \theta$, and K^{i+1} is obtained from K^i by applying a transformation that depends on the innermost temporal operator in θ^i, which is replaced by a fresh atomic proposition in θ^{i+1}. We show that a K^i-prob-bad-prefix for θ^i can be constructed by extending a K^{i+1}-prob-bad-prefix for θ^{i+1}, resulting in a recursive construction of a K-prob-bad-prefix for θ.

▶ **Theorem 21.** *Finding a K-prob-bad-prefix for a K-prob-counterable LTL formula θ can be done in time $O(|K| \cdot 2^{|\theta|})$ or in space polynomial in $|\theta|$ and polylogarithmic in $|K|$. Furthermore, the K-prob-bad-prefix that is found is of length $O(|K| \cdot 2^{|\theta|})$.*

We now study the length of shortest K-prob-bad-prefixes.

▶ **Theorem 22.** *The length of a shortest K-prob-bad-prefix for a K-prob-counterable language given by an LTL formula ψ is tightly exponential in $|\psi|$ and tightly linear in $|K|$.*

Note that the K-prob-bad-prefix that our algorithm finds is not necessarily the shortest, however it matches the lower bound from Theorem 22.

Thus, we showed that the probabilistic approach for relative bad prefixes for LTL formulas is exponentially better than the non-probabilistic approach both in its complexity and in the length of the prefixes.

8 On Probabilistic Counterability

In this section we study prob-counterability. The solutions to the three basic problems are specified in Theorems 23, 24, and 25 below. The upper bound for the first follows from the fact that, by Theorem 2, an ω-regular language L is prob-counterable iff $Pr[L] < 1$, which,

by [11], can be checked in PSPACE. The latter two follow from the results in Section 7, taking the Kripke structure to be K_{AP}.

▶ **Theorem 23.** *The problem of deciding whether a language L is prob-counterable is PSPACE-complete for L given by an LTL formula or by an NBW.*

▶ **Theorem 24.** *Let $\mathcal{A}$ be an NBW. Finding a shortest prob-bad-prefix for $L(\mathcal{A})$ can be done in time exponential in $|\mathcal{A}|$, or in space polynomial in $|\mathcal{A}|$. Furthermore, the length of the shortest prob-bad-prefix is tightly exponential in $|\mathcal{A}|$.*

▶ **Theorem 25.** *Finding a prob-bad-prefix for a prob-counterable LTL formula ψ can be done in time $2^{O(|\psi|)}$ or in space polynomial in $|\psi|$. Furthermore, the prob-bad-prefix that is found is of length $2^{O(|\psi|)}$. The shortest prob-bad-prefix for ψ is tightly exponential in $|\psi|$.*

Thus, the exponential advantage of the probabilistic approach in the case the language is given by an LTL formula is carried over to the non-relative setting. When the specification is given by means of an NBW, the complexities of the probabilistic and non-probabilistic approaches coincide. The probabilistic approach, however, may return more bad prefixes.

9 Discussion

We extended the applicability of finite counterexamples by introducing relative and probabilistic bad-prefixes. This lifts the advantage of safety properties, which always have bad-prefixes, to ω-regular languages that are not safety. We believe that K-bad-prefixes and K-prob-bad-prefixes may be very helpful in practice, as they describe a finite execution that leads the system to an error state. From a computational point of view, finding a K-bad-prefix for an LTL formula ψ is unfortunately EXPSPACE-complete in $|\psi|$. Experience shows that even highly complex algorithms often run surprisingly well in practice. Also here, the complexity originates from the blow-up in the translation of LTL to automata, which rarely happens in practice. In cases the complexity is too high, we suggest the following two alternatives, which do not go beyond the PSPACE complexity of LTL model checking (and, like model checking, are NLOGSPACE in K): (1) Recall that when ψ is K-safety and $K \not\models \psi$, then finding a K-bad-prefix can be done in PSPACE. Thus, we suggest to check ψ for K-safety with the algorithm from Theorem 13, and then apply the algorithm from Theorem 16. (2) Recall that finding a K-prob-bad-prefix is only PSPACE-complete. Thus, we suggest to apply the algorithm from Theorem 21. Note that the probabilistic approach is not only exponentially less complex, but may be essential when ψ is K-prob-counterable and not K-counterable.

When a user gets a lasso-shaped counterexample, he can verify that indeed it does not satisfy the specification. For finite bad-prefixes, the user knows that they lead the system to an error state, and it is desirable to accompany the prefix with information explaining why these states are erroneous. We suggest the following three types of explanations. (1) A K-bad-prefix leads the product $K \times \mathcal{A}_\psi$ to states $\langle w, S\rangle$ that are empty. Recall that the states of $\mathcal{A}_\psi$ consist of subsets of subformulas of ψ, and that $\langle w, S\rangle$ being empty means that w does not satisfy the conjunction of the formulas in S [37]. Returning S to the user explains what makes w an error state. (2) Researchers have studied *certified model checking* [24], where a positive answer of model checking (that is, $K \models \psi$) is accompanied by a certificate – a compact explanation as to why $K \times \mathcal{A}_{\neg\psi}$ is empty. In our setting, certificates can provide a compact explanation as to why $K \times \mathcal{A}_\psi$ with initial state $\langle w, S\rangle$ is empty. (3) When a K-prob-bad-prefix u that is not a K-bad-prefix is returned, it may be helpful to accompany u with an infinite lasso-shaped computation τ of the system that starts with u and does satisfy

the specification. Thus, the user would get an *exception*: he would know that almost all computations that start in u except for τ (and possibly more computations, whose probability is 0) violate ψ. The exceptional correct behavior would help the user understand why almost all other behaviors are incorrect.

Acknowledgment. We thank Moshe Y. Vardi for helpful discussions.

References

1 E. Ábrahám, B. Becker, C. Dehnert, N. Jansen, J.-P. Katoen, and R. Wimmer. Counterexample generation for discrete-time Markov models. In *FMESM*, pages 65–121. Springer, 2014.

2 B. Alpern and F. B. Schneider. Defining liveness. *IPL*, 21:181–185, 1985.

3 B. Alpern and F. B. Schneider. Recognizing safety and liveness. *Dist. computing*, 2:117–126, 1987.

4 D. A. Basin, C. C. Jiménez, F. Klaedtke, and E. Zalinescu. Deciding safety and liveness in TPTL. *IPL*, 114(12):680–688, 2014.

5 A. Biere, C. Artho, and V. Schuppan. Liveness checking as safety checking. In *Proc. 7th FMICS*, LNTCS 66:2, 2002.

6 J. R. Büchi. On a decision method in restricted second order arithmetic. In *Proc. Int. Congress on Logic, Method, and Philosophy of Science. 1960*, pages 1–12. Stanford University Press, 1962.

7 E. M. Clarke, A. Biere, R. Raimi, and Y. Zhu. Bounded model checking using satisfiability solving. *Formal Methods in System Design*, 19(1):7–34, 2001.

8 E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *Proc. 12th CAV*, LNCS 1855, pages 154–169. Springer, 2000.

9 E. M. Clarke, O. Grumberg, K. L. McMillan, and X. Zhao. Efficient generation of counterexamples and witnesses in symbolic model checking. In *Proc. 32st DAC*, pages 427–432, 1995.

10 T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press and McGraw-Hill, 1990.

11 C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *J. ACM*, 42:857–907, 1995.

12 J. M. Couvreur, N. Saheb, and G. Sutre. An optimal automata approach to LTL model checking of probabilistic systems. In *Proc. 10th LPAR*, LNCS 2850, pages 361–375. Springer, 2003.

13 S. Ben David and O. Kupferman. A framework for ranking vacuity results. In *Proc. 11th ATVA*, LNCS 8172, pages 148–162. Springer, 2013.

14 V. Diekert, A. Muscholl, and I. Walukiewicz. A Note on Monitors and Büchi automata In *Proc. 12th ICTAC*, to appear, 2015.

15 D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the temporal analysis of fairness. In *Proc. 7th POPL*, pages 163–173, 1980.

16 C. Grinstead and J. Laurie Snell. 11:markov chains. In *Introduction to Probability*. American Mathematical Society, 1997.

17 K. Havelund and G. Rosu. Efficient monitoring of safety properties. *STTT*, 6(2):18–173, 2004.

18 T. A. Henzinger. Sooner is safer than later. *IPL*, 43(3):135–141, 1992.

19 J.-P. Katoen, L. Song, and L. Zhang. Probably safe or live. In *Proc. 29th LICS*, pages 55:1–55:10, 2014.

20 J. G. Kemeny, J. L. Snell, and A. W. Knapp. *Denumerable Markov Chains*. Springer, 1976.

21 O. Kupferman and R. Lampert. On the construction of fine automata for safety properties. In *Proc. 4th ATVA*, LNCS 4218, pages 110–124. Springer, 2006.

22 O. Kupferman and S. Sheinvald-Faragy. Finding shortest witnesses to the nonemptiness of automata on infinite words. In *Proc. 17th CONCUR*, LNCS 4137, pages 492–508. Springer, 2006.

23 O. Kupferman and M. Y. Vardi. Model checking of safety properties. *Formal Methods in System Design*, 19(3):291–314, 2001.

24 O. Kupferman and M. Y. Vardi. From complementation to certification. In *Proc. 10th TACAS*, LNCS 2988, pages 591–606. Springer, 2004.

25 O. Kupferman and M. Y. Vardi. Synthesis of trigger properties. In *Proc. 16th LPAR*, LNCS 6355, pages 312–331. Springer, 2010.

26 M. Lippmann. Temporalised description logics for monitoring partially observable events. 2014.

27 Z. Manna and A. Pnueli. Adequate proof principles for invariance and liveness properties of concurrent programs. *Science of Computer Programming*, 4:257–289, 1984.

28 A. R. Meyer and L. J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential time. In *Proc. 13th SWAT*, pages 125–129, 1972.

29 U. Nitsche and P. Wolper. Relative liveness and behavior abstraction. In *Proc. 24th POPL*, pages 45–52. ACM, 1997.

30 S. Owicki and L. Lamport. Proving liveness properties of concurrent programs. *ACM TOPLAS*, 4(3):455–495, 1982.

31 N. Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. *LMCS*, 3(3):5, 2007.

32 A. Pnueli. The temporal semantics of concurrent programs. *TCS*, 13:45–60, 1981.

33 V. Schuppan and A. Biere. Shortest counterexamples for symbolic model checking of LTL with past. In *Proc. 11th TACAS*, LNCS 3440, pages 493–509. Springer, 2005.

34 V. Schuppan and A. Biere. Liveness checking as safety checking for infinite state spaces. *ENTCS*, 149(1):79–96, 2006.

35 A. P. Sistla. Safety, liveness and fairness in temporal logic. *FAC*, 6:495–511, 1994.

36 M. Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *Proc. 26th FOCS*, pages 327–338, 1985.

37 M. Y. Vardi and P. Wolper. Reasoning about infinite computations. *I& C*, 115(1):1–37, 1994.

A Model Checking Procedure for Interval Temporal Logics based on Track Representatives

Alberto Molinari[1], Angelo Montanari[1], and Adriano Peron[2]

1 **Department of Mathematics and Computer Science**
University of Udine, Italy
molinari.alberto@gmail.com; angelo.montanari@uniud.it

2 **Department of Electrical Engineering and Information Technology**
University of Napoli Federico II, Italy
adrperon@unina.it

Abstract

Model checking is commonly recognized as one of the most effective tools for system verification. While it has been systematically investigated in the context of classical, point-based temporal logics, it is still largely unexplored in the interval logic setting. Recently, a non-elementary model checking algorithm for Halpern and Shoham's modal logic of time intervals HS, interpreted over finite Kripke structures, has been proposed, together with a proof of the EXPSPACE-hardness of the problem. In this paper, we devise an EXPSPACE model checking procedure for two meaningful HS fragments. It exploits a suitable contraction technique that allows one to replace sufficiently long tracks of a Kripke structure by equivalent shorter ones.

1998 ACM Subject Classification D.2.4 Software/Program Verification

Keywords and phrases Interval Temporal Logic, Model Checking, Complexity

Digital Object Identifier 10.4230/LIPIcs.CSL.2015.193

1 Introduction

Given a formal specification of the desired properties of a system and a model of its behaviour, model checking algorithms allow one to verify the former against the latter [6]. While the model checking problem has been systematically investigated in the context of classical, point-based temporal logics, it is still largely unexplored in the interval logic setting.

Interval temporal logic (ITL) has been proposed as a more expressive formalism for temporal representation and reasoning than standard point-based one [9, 24]. On the positive side, expressiveness of ITLs makes them well suited for a number of applications in a variety of fields, including formal verification, computational linguistics, and planning, e.g., [20, 22]. On the negative side, in most cases their satisfiability problem turns out to be undecidable, and, in the few cases of decidable ITLs, the standard proof machinery, like Rabin's theorem, is usually not applicable.

A prominent position among ITLs is occupied by Halpern and Shoham's modal logic of time intervals (HS, for short) [9]. HS features one modality for each of the 13 possible ordering relations between pairs of intervals (the so-called Allen's relations [1]), apart from the equality relation. In [9], it has been shown that the satisfiability problem for HS interpreted over all relevant (classes of) linear orders is highly undecidable. Since then, a lot of work has been done on the satisfiability problem for HS fragments, which has shown that undecidability prevails over them (see [2] for an up-to-date account of undecidable fragments). However,

24th EACSL Annual Conference on Computer Science Logic (CSL 2015).
Editor: Stephan Kreutzer; pp. 193–210

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

meaningful exceptions exist, including the interval logic of temporal neighbourhood and the interval logic of sub-intervals [3, 4, 5, 19].

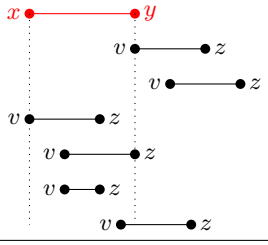
In this paper, we focus our attention on the model checking problem for ITLs. Unlike the case of satisfiability checking, little work has been done on model checking [13, 14, 16, 18] (it is worth pointing out that, in contrast to the case of point-based, linear temporal logics, there is not an easy reduction from the model checking problem to the validity/satisfiability one). In the classical formulation of the model checking problem [6], systems are usually modelled as (finite) labelled state-transition graphs (Kripke structures), and point-based temporal logics are used to analyse, for each path/track in a Kripke structure, how proposition letters labelling the states change from one state to the next one along the path. To check interval properties of computations, one needs to collect information about states into computation stretches. This amounts to interpreting each finite path of a Kripke structure as an interval, and to suitably defining its labelling on the basis of the labelling of the states that compose it.

In [13, 14], Lomuscio and Michaliszyn address the model checking problem for epistemic extensions of some HS fragments. In [13], they focus their attention on the fragment $HS[B, E, D]$ of Allen's relations *started-by*, *finished-by*, and *contains* extended with epistemic modalities. They consider a restricted form of model checking which verifies the given specification against a single (finite) initial computation interval (*not* all possible initial computation intervals), and prove that it is a PSPACE-complete problem. Moreover, they show that the problem for the purely temporal fragment of the logic is in PTIME. In [14], they show that the picture drastically changes with other HS fragments that allow one to access infinitely many tracks/intervals. In particular, they prove that the model checking problem for the fragment $HS[A, \overline{B}, L]$ of Allen's relations *meets*, *starts*, and *before*, extended with epistemic modalities, is decidable with a non-elementary upper bound.

In [16, 18], Montanari et al. outline a general characterization of the model checking problem for full HS, interpreted over finite Kripke structures (under the homogeneity assumption [23]). Their semantic assumptions differ from those made in [13], making it difficult to compare the two research contributions. In both cases, formulas of ITL are evaluated over finite paths/tracks obtained from the unravelling of a finite Kripke structure. However, in [18] a proposition letter holds over an interval (track) if and only if it holds over all its states (homogeneity principle), while in [13] truth of proposition letters is defined over pairs of states (the endpoints of tracks/intervals). In [18], the authors introduce the basic elements of the picture, namely, the interpretation of HS formulas over (abstract) interval models, the mapping of finite Kripke structures into (abstract) interval models, the notion of track descriptor, and a small model theorem proving (with a non-elementary procedure) the decidability of the model checking problem for full HS against finite Kripke structures. However, technical details of the proofs are not fully worked out and no lower bound to the complexity of the problem, that is, no hardness result, is given. In addition, they outline a PSPACE model checking procedure for two HS fragments, but it turns out to be flawed. In [16], Molinari et al. work out the model checking problem for full HS in all its details, and prove that it is EXPSPACE-hard.

In this paper, we prove that the model checking problem for two large HS fragments, namely, the fragment $HS[A, \overline{A}, B, \overline{B}, \overline{E}]$ of Allen's relations *meets*, *met-by*, *started-by*, *starts* and *finishes*, and the fragment $HS[A, \overline{A}, E, \overline{B}, \overline{E}]$ of Allen's relations *meets*, *met-by*, *finished-by*, *starts* and *finishes*, is in EXPSPACE. Moreover, we prove that it is NEXP-hard, provided that a succinct encoding of formulas is used (otherwise, we can only give an NP-hardness result).

Table 1 Allen's interval relations and corresponding HS modalities.

Allen's relation	HS	Definition w.r.t. interval structures	Example
MEETS	$\langle A \rangle$	$[x,y]\mathcal{R}_A[v,z] \iff y = v$	
BEFORE	$\langle L \rangle$	$[x,y]\mathcal{R}_L[v,z] \iff y < v$	
STARTED-BY	$\langle B \rangle$	$[x,y]\mathcal{R}_B[v,z] \iff x = v \wedge z < y$	
FINISHED-BY	$\langle E \rangle$	$[x,y]\mathcal{R}_E[v,z] \iff y = z \wedge x < v$	
CONTAINS	$\langle D \rangle$	$[x,y]\mathcal{R}_D[v,z] \iff x < v \wedge z < y$	
OVERLAPS	$\langle O \rangle$	$[x,y]\mathcal{R}_O[v,z] \iff x < v < y < z$	

The paper is organized as follows. In Section 2 we provide some background knowledge. In Section 3 we introduce the key notion of descriptor sequence for a track of a finite Kripke structure, and we exploit it to define an indistinguishability (equivalence) relation over tracks. In Section 4 we prove a small model theorem, showing that we can select a track representative of bounded length from each equivalence class, we outline a model checking procedure, and we provide a lower bound to the complexity of the problem. Conclusions give a short assessment of the work done and describe future research directions. Due to space limitations, all proofs are omitted; they can be found in [17].

2 Background Knowledge

2.1 The interval temporal logic HS

An interval algebra to reason about intervals and their relative order was first proposed by Allen [1]; then, a systematic logical study of ITLs was done by Halpern and Shoham, who introduced the logic HS featuring one modality for each Allen's interval relation [9], except for equality. Table 1 depicts 6 of the 13 Allen's relations together with the corresponding HS (existential) modalities. The other 7 are equality and the 6 inverse relations (given a binary relation $\mathcal{R}$, the inverse relation $\overline{\mathcal{R}}$ is such that $b\overline{\mathcal{R}}a$ if and only if $a\mathcal{R}b$).

The language of HS features a set of proposition letters $\mathcal{AP}$, the Boolean connectives $\neg$ and $\wedge$, and a temporal modality for each of the (non trivial) Allen's relations, namely, $\langle A \rangle$, $\langle L \rangle$, $\langle B \rangle$, $\langle E \rangle$, $\langle D \rangle$, $\langle O \rangle$, $\langle \overline{A} \rangle$, $\langle \overline{L} \rangle$, $\langle \overline{B} \rangle$, $\langle \overline{E} \rangle$, $\langle \overline{D} \rangle$ and $\langle \overline{O} \rangle$. HS formulas are defined as follows:

$$\psi ::= p \mid \neg\psi \mid \psi \wedge \psi \mid \langle X \rangle \psi \mid \langle \overline{X} \rangle \psi, \quad \text{with } p \in \mathcal{AP},\ X \in \{A, L, B, E, D, O\}.$$

We will make use of the standard abbreviations of propositional logic. Moreover, for all X, dual universal modalities $[X]\psi$ and $[\overline{X}]\psi$ are respectively defined as $\neg\langle X \rangle \neg\psi$ and $\neg\langle \overline{X} \rangle \neg\psi$.

We will assume the *strict semantics* of HS: only intervals made of at least two points are allowed.[1] All HS modalities can be expressed in terms of $\langle A \rangle$, $\langle B \rangle$, and $\langle E \rangle$, and the transposed modalities $\langle \overline{A} \rangle, \langle \overline{B} \rangle$, and $\langle \overline{E} \rangle$ as follows: $\langle L \rangle \psi \equiv \langle A \rangle \langle A \rangle \psi$, $\langle \overline{L} \rangle \psi \equiv \langle \overline{A} \rangle \langle \overline{A} \rangle \psi$, $\langle D \rangle \psi \equiv \langle B \rangle \langle E \rangle \psi$, $\langle O \rangle \psi \equiv \langle E \rangle \langle \overline{B} \rangle \psi$, $\langle \overline{D} \rangle \psi \equiv \langle \overline{B} \rangle \langle \overline{E} \rangle \psi$, and $\langle \overline{O} \rangle \psi \equiv \langle B \rangle \langle \overline{E} \rangle \psi$.

Given any subset of Allen's relations $\{X_1, \cdots, X_n\}$, we denote by $HS[X_1, \cdots, X_n]$ the fragment of HS that features modalities $X_1, \cdots, X_n$ only.

[1] HS modalities are *mutually exclusive* and *jointly exhaustive* only in the strict semantics, i.e., exactly one of them holds between any two intervals. However, the strict semantics can easily be "relaxed" to include point intervals, and all results we are going to prove hold for the non-strict semantics as well.

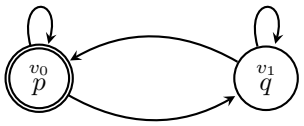


Figure 1 The Kripke structure $\mathcal{K}_{Equiv}$.

HS can be viewed as a multi-modal logic with the 6 primitive modalities $\langle \mathrm{A} \rangle$, $\langle \mathrm{B} \rangle$, $\langle \mathrm{E} \rangle$, $\langle \overline{\mathrm{A}} \rangle$, $\langle \overline{\mathrm{B}} \rangle$, and $\langle \overline{\mathrm{E}} \rangle$. Accordingly, HS semantics can be defined over a multi-modal Kripke structure, called here an *abstract interval model*, in which (strict) intervals are treated as atomic objects and Allen's relations as simple binary relations between pairs of them.

▶ **Definition 1** ([16]). An *abstract interval model* is a tuple $\mathcal{A} = (\mathcal{AP}, \mathbb{I}, A_{\mathbb{I}}, B_{\mathbb{I}}, E_{\mathbb{I}}, \sigma)$, where $\mathcal{AP}$ is a finite set of proposition letters, $\mathbb{I}$ is a possibly infinite set of atomic objects (worlds), $A_{\mathbb{I}}$, $B_{\mathbb{I}}$, $E_{\mathbb{I}}$ are three binary relations over $\mathbb{I}$ and $\sigma : \mathbb{I} \mapsto 2^{\mathcal{AP}}$ is a (total) labeling function which assigns a set of proposition letters to each world.

Intuitively, in the interval setting, $\mathbb{I}$ is a set of intervals, $A_{\mathbb{I}}$, $B_{\mathbb{I}}$, and $E_{\mathbb{I}}$ are interpreted as Allen's interval relations A (*meets*), B (*started-by*), and E (*finished-by*), respectively, and σ assigns to each interval the set of proposition letters that hold over it.

Given an abstract interval model $\mathcal{A} = (\mathcal{AP}, \mathbb{I}, A_{\mathbb{I}}, B_{\mathbb{I}}, E_{\mathbb{I}}, \sigma)$ and an interval $I \in \mathbb{I}$, the truth of an HS formula over I is defined by structural induction on the formula as follows:

(i) $\mathcal{A}, I \models p$ iff $p \in \sigma(I)$, for any proposition letter $p \in \mathcal{AP}$;
(ii) $\mathcal{A}, I \models \neg\psi$ iff it is not true that $\mathcal{A}, I \models \psi$;
(iii) $\mathcal{A}, I \models \psi \wedge \phi$ iff $\mathcal{A}, I \models \psi$ and $\mathcal{A}, I \models \phi$;
(iv) $\mathcal{A}, I \models \langle X \rangle \psi$, for $X \in \{A, B, E\}$, iff there exists $J \in \mathbb{I}$ such that $I \, X_{\mathbb{I}} \, J$ and $\mathcal{A}, J \models \psi$;
(v) $\mathcal{A}, I \models \langle \overline{X} \rangle \psi$, for $\overline{X} \in \{\overline{A}, \overline{B}, \overline{E}\}$, iff there exists $J \in \mathbb{I}$ such that $J \, X_{\mathbb{I}} \, I$ and $\mathcal{A}, J \models \psi$.

2.2 Kripke structures and abstract interval models

In this section, we define a mapping from Kripke structures to abstract interval models that makes it possible to specify properties of systems by means of HS formulas.

▶ **Definition 2.** A finite Kripke structure $\mathcal{K}$ is a tuple $(\mathcal{AP}, W, \delta, \mu, w_0)$, where $\mathcal{AP}$ is a set of proposition letters, W is a finite set of states, $\delta \subseteq W \times W$ is a left-total relation between pairs of states, $\mu : W \mapsto 2^{\mathcal{AP}}$ is a total labelling function, and $w_0 \in W$ is the initial state.

For all $w \in W$, $\mu(w)$ is the set of proposition letters which hold at that state, while δ is the transition relation which constrains the evolution of the system over time.

Figure 1 depicts a Kripke structure, $\mathcal{K}_{Equiv}$, with two states (the initial state is identified by a double circle). Formally, $\mathcal{K}_{Equiv}$ is defined by the following quintuple:

$$(\{p, q\}, \{v_0, v_1\}, \{(v_0, v_0), (v_0, v_1), (v_1, v_0), (v_1, v_1)\}, \mu, v_0),$$

where $\mu(v_0) = \{p\}$ and $\mu(v_1) = \{q\}$.

▶ **Definition 3.** A track ρ over a finite Kripke structure $\mathcal{K} = (\mathcal{AP}, W, \delta, \mu, w_0)$ is a *finite* sequence of states $v_0 \cdots v_n$, with $n \geq 1$, such that for all $i \in \{0, \cdots, n-1\}$, $(v_i, v_{i+1}) \in \delta$.

Let $\mathrm{Trk}_{\mathcal{K}}$ be the (infinite) set of all tracks over a finite Kripke structure $\mathcal{K}$. For any track $\rho = v_0 \cdots v_n \in \mathrm{Trk}_{\mathcal{K}}$, we define: $|\rho| = n + 1$, $\rho(i) = v_i$, $\mathrm{states}(\rho) = \{v_0, \cdots, v_n\} \subseteq W$, $\mathrm{intstates}(\rho) = \{v_1, \cdots, v_{n-1}\} \subseteq W$, $\mathrm{fst}(\rho) = v_0$ and $\mathrm{lst}(\rho) = v_n$; moreover $\rho(i, j) = v_i \cdots v_j$

is a subtrack of ρ for $0 \leq i < j \leq |\rho| - 1$. Finally, $\mathrm{Pref}(\rho) = \{\rho(0, i) \mid 1 \leq i \leq |\rho| - 2\}$ is the set of all proper prefixes of ρ, and $\mathrm{Suff}(\rho) = \{\rho(i, |\rho| - 1) \mid 1 \leq i \leq |\rho| - 2\}$ is the set of all proper suffixes of ρ. Notice that the length of tracks, prefixes, and suffixes is greater than 1, as they will be mapped into strict intervals. If $\mathrm{fst}(\rho) = w_0$, ρ is said to be an *initial track*. In the following, we will denote by $\rho \cdot \rho'$ the concatenation of the tracks ρ and ρ', and by ρ^n the track obtained by concatenating n copies of ρ.

An abstract interval model (over $\mathrm{Trk}_{\mathcal{K}}$) can be naturally associated with a finite Kripke structure by interpreting every track as an interval bounded by its first and last states.

▶ **Definition 4** ([16]). The abstract interval model induced by a finite Kripke structure $\mathcal{K} = (\mathcal{AP}, W, \delta, \mu, w_0)$ is the abstract interval model $\mathcal{A}_{\mathcal{K}} = (\mathcal{AP}, \mathbb{I}, A_{\mathbb{I}}, B_{\mathbb{I}}, E_{\mathbb{I}}, \sigma)$, where $\mathbb{I} = \mathrm{Trk}_{\mathcal{K}}$, $A_{\mathbb{I}} = \{(\rho, \rho') \in \mathbb{I} \times \mathbb{I} \mid \mathrm{lst}(\rho) = \mathrm{fst}(\rho')\}$, $B_{\mathbb{I}} = \{(\rho, \rho') \in \mathbb{I} \times \mathbb{I} \mid \rho' \in \mathrm{Pref}(\rho)\}$, $E_{\mathbb{I}} = \{(\rho, \rho') \in \mathbb{I} \times \mathbb{I} \mid \rho' \in \mathrm{Suff}(\rho)\}$, and $\sigma : \mathbb{I} \mapsto 2^{\mathcal{AP}}$ with $\sigma(\rho) = \bigcap_{w \in \mathrm{states}(\rho)} \mu(w)$ for all $\rho \in \mathbb{I}$.

In Definition 4, relations $A_{\mathbb{I}}$, $B_{\mathbb{I}}$, and $E_{\mathbb{I}}$ are interpreted as Allen's interval relations A, B, and E, respectively. Moreover, according to the definition of σ, a proposition letter $p \in \mathcal{AP}$ holds over $\rho = v_0 \cdots v_n$ if and only if it holds over all the states $v_0, \ldots, v_n$ of ρ. This conforms to the *homogeneity principle*, according to which a proposition letter holds over an interval if and only if it holds over all of its subintervals.

Satisfiability of an HS formula over a finite Kripke structure can be given in terms of induced abstract interval models.

▶ **Definition 5** (Satisfiability of HS formulas over Kripke structures). Let $\mathcal{K}$ be a finite Kripke structure, ρ be a track in $\mathrm{Trk}_{\mathcal{K}}$, and ψ be an HS formula. We say that the pair $(\mathcal{K}, \rho)$ satisfies ψ, denoted by $\mathcal{K}, \rho \models \psi$, if and only if it holds that $\mathcal{A}_{\mathcal{K}}, \rho \models \psi$.

The *model checking problem* for HS over finite Kripke structures is the problem of deciding whether $\mathcal{K} \models \psi$.

▶ **Definition 6.** Let $\mathcal{K}$ be a finite Kripke structure and ψ be an HS formula. We say that $\mathcal{K}$ models ψ, denoted by $\mathcal{K} \models \psi$, if and only if for all *initial* tracks $\rho \in \mathrm{Trk}_{\mathcal{K}}$, it holds that $\mathcal{K}, \rho \models \psi$.

Some meaningful properties of tracks that are expressible in HS can be found in [16]. For instance, the formula $[B]\bot$ can be used to select all and only the tracks of length 2. Indeed, given any ρ with $|\rho| = 2$, independently of $\mathcal{K}$, it holds that $\mathcal{K}, \rho \models [B]\bot$, because ρ has no (strict) prefixes. On the other hand, it holds that $\mathcal{K}, \rho \models \langle \mathrm{B} \rangle \top$ if (and only if) $|\rho| > 2$. Let $\ell(k)$ be a shorthand for $[B]^{k-1}\bot \wedge \langle \mathrm{B} \rangle^{k-2} \top$: it holds that $\mathcal{K}, \rho \models \ell(k)$ if and only if $|\rho| = k$.

2.3 The notion of B_k-descriptor

For any finite Kripke structure $\mathcal{K}$, one can find a corresponding induced abstract interval model $\mathcal{A}_{\mathcal{K}}$, featuring one interval for each track of $\mathcal{K}$. Since $\mathcal{K}$ has loops (each state must have at least one successor), the number of its tracks, and thus the number of intervals of $\mathcal{A}_{\mathcal{K}}$, is infinite. In [16], given a finite Kripke structure and an HS formula φ, the authors show how to obtain a *finite* representation for each (possibly infinite) set of tracks which are equivalent with respect to satisfiability of HS formulas of the same structural complexity as φ. By making use of such a representation, they prove that the model checking problem for (full) HS is decidable (with a non-elementary upper bound) and it is EXPSPACE-hard if a suitable encoding of HS formulas is exploited [16]. In this paper, we restrict our attention to $HS[A, \overline{A}, B, \overline{B}, \overline{E}]$ (and the symmetric $HS[A, \overline{A}, E, \overline{B}, \overline{E}]$) and we provide a lower complexity model checking algorithm for it. We start with the definition of some basic notions.

▶ **Definition 7.** Let ψ be an $HS[A, \overline{A}, B, \overline{B}, \overline{E}]$ formula. The B-nesting depth of ψ, denoted by $\mathrm{Nest_B}(\psi)$, is defined by induction on the complexity of the formula as follows:

(i) $\mathrm{Nest_B}(p) = 0$, for any proposition letter $p \in \mathcal{AP}$;
(ii) $\mathrm{Nest_B}(\neg\psi) = \mathrm{Nest_B}(\psi)$;
(iii) $\mathrm{Nest_B}(\psi \wedge \phi) = \max\{\mathrm{Nest_B}(\psi), \mathrm{Nest_B}(\phi)\}$;
(iv) $\mathrm{Nest_B}(\langle \mathrm{B} \rangle \psi) = 1 + \mathrm{Nest_B}(\psi)$;
(v) $\mathrm{Nest_B}(\langle \mathrm{X} \rangle \psi) = \mathrm{Nest_B}(\psi)$, for $X \in \{A, \overline{A}, \overline{B}, \overline{E}\}$.

Making use of Definition 7, we can introduce a relation of k-equivalence over tracks.

▶ **Definition 8.** Let $\mathcal{K}$ be a finite Kripke structure and ρ and ρ' be two tracks in $\mathrm{Trk}_{\mathcal{K}}$. We say that ρ and ρ' are k-equivalent if and only if, for every $HS[A, \overline{A}, B, \overline{B}, \overline{E}]$ formula ψ with $\mathrm{Nest_B}(\psi) = k$, $\mathcal{K}, \rho \models \psi$ if and only if $\mathcal{K}, \rho' \models \psi$.

It can be easily proved that k-equivalence propagates downwards.

▶ **Proposition 9.** *Let $\mathcal{K}$ be a finite Kripke structure and ρ and ρ' be two tracks in* $\mathrm{Trk}_{\mathcal{K}}$. *If ρ and ρ' are k-equivalent, then they are h-equivalent, for all $0 \leq h \leq k$.*

We are now ready to define the key notion of *descriptor* for a track of a Kripke structure.

▶ **Definition 10** ([16])**.** Let $\mathcal{K} = (\mathcal{AP}, W, \delta, \mu, v_0)$ be a finite Kripke structure, $\rho \in \mathrm{Trk}_{\mathcal{K}}$, and $k \in \mathbb{N}$. The B_k-descriptor for ρ is a labelled tree $\mathcal{D} = (V, E, \lambda)$ of depth k, where V is a finite set of vertices, $E \subseteq V \times V$ is a set of edges, and $\lambda : V \mapsto W \times 2^W \times W$ is a node labelling function, inductively defined as follows:

- for $k = 0$, the B_k-descriptor for ρ is the tree $\mathcal{D} = (\mathrm{root}(\mathcal{D}), \emptyset, \lambda)$, where $\lambda(\mathrm{root}(\mathcal{D})) = (\mathrm{fst}(\rho), \mathrm{intstates}(\rho), \mathrm{lst}(\rho))$;
- for $k > 0$, the B_k-descriptor for ρ is the tree $\mathcal{D} = (V, E, \lambda)$, where $\lambda(\mathrm{root}(\mathcal{D})) = (\mathrm{fst}(\rho), \mathrm{intstates}(\rho), \mathrm{lst}(\rho))$, which satisfies the following conditions:
  1. for each prefix ρ' of ρ, there exists $v \in V$ such that $(\mathrm{root}(\mathcal{D}), v) \in E$ and the subtree rooted in v is the B_{k-1}-descriptor for ρ';
  2. for each vertex $v \in V$ such that $(\mathrm{root}(\mathcal{D}), v) \in E$, there exists a prefix ρ' of ρ such that the subtree rooted in v is the B_{k-1}-descriptor for ρ';
  3. for all pairs of edges $(\mathrm{root}(\mathcal{D}), v'), (\mathrm{root}(\mathcal{D}), v'') \in E$, if the subtree rooted in v' is isomorphic to the subtree rooted in v'', then $v' = v''$ (here and in the following, we write subtree for maximal subtree).

Condition 3 of Definition 10 simply states that no two subtrees whose roots are siblings can be isomorphic. A B_0-descriptor $\mathcal{D}$ for a track consists of its root only, which is denoted by $\mathrm{root}(\mathcal{D})$. A label of a node will be referred to as a *descriptor element*.

Basically, for any $k \geq 0$, the label of the root of the B_k-descriptor $\mathcal{D}$ for ρ is the triple $(\mathrm{fst}(\rho), \mathrm{intstates}(\rho), \mathrm{lst}(\rho))$. Each prefix ρ' of ρ is associated with some subtree whose root is labelled with $(\mathrm{fst}(\rho'), \mathrm{intstates}(\rho'), \mathrm{lst}(\rho'))$ and is a child of the root of $\mathcal{D}$. Such a construction is then iteratively applied to the children of the root until either depth k is reached or a track of length 2 is being considered on a node.

Hereafter, two descriptors will be considered *equal up to isomorphism*.

As an example, in Figure 2 we show the B_2-descriptor for the track $\rho = v_0v_1v_0v_0v_0v_0v_1$ of $\mathcal{K}_{Equiv}$ (Figure 1). It is worth noticing that there exist two distinct prefixes of ρ, that is, the tracks $\rho' = v_0v_1v_0v_0v_0v_0$ and $\rho'' = v_0v_1v_0v_0v_0$, which have the same B_1-descriptor. Since, according to Definition 10, no tree can occur more than once as a subtree of the same node (in this example, the root), in the B_2-descriptor for ρ prefixes ρ' and ρ'' are represented

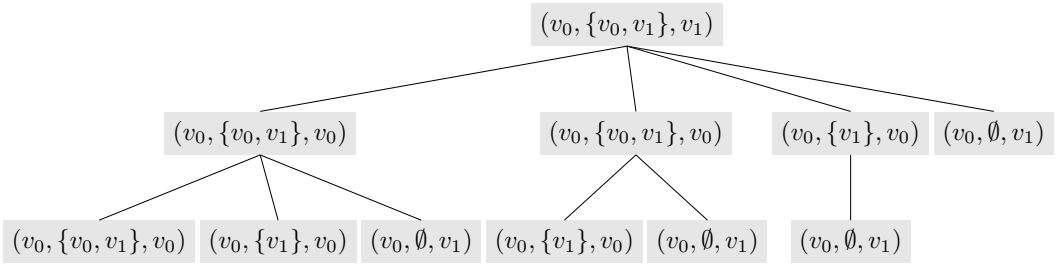


Figure 2 The B_2-descriptor for the track $v_0v_1v_0v_0v_0v_0v_1$ of $\mathcal{K}_{Equiv}$.

by the same tree (the first subtree of the root on the left). In general, it holds that the root of a descriptor for a track with h proper prefixes does not necessarily have h children.

In general, B-descriptors do not convey enough information to determine which track they were built from; however, they can be exploited to determine which $HS[A, \overline{A}, B, \overline{B}, \overline{E}]$ formulas are satisfied by the track from which they were built.

In [16], the authors prove that, for a finite Kripke structure $\mathcal{K}$, there is a *finite number* (non-elementary w.r.t. $|W|$ and k) of possible B_k-descriptors; moreover the number of nodes of a descriptor has a non-elementary upper bound as well. Since the number of tracks of $\mathcal{K}$ is infinite, and for any $k \in \mathbb{N}$ the set of B_k-descriptors for its tracks is finite, at least one B_k-descriptor must be the B_k-descriptor of *infinitely many* tracks; thus B_k-descriptors naturally induce an equivalence relation of finite index over the set of tracks of a finite Kripke structure (*k-descriptor equivalence relation*).

▶ **Definition 11.** Let $\mathcal{K}$ be a finite Kripke structure, $\rho, \rho' \in \mathrm{Trk}_{\mathcal{K}}$, and $k \in \mathbb{N}$. We say that ρ and ρ' are k-descriptor equivalent ($\rho \sim_k \rho'$) iff the B_k-descriptors for ρ and ρ' coincide.

The following lemma holds.

▶ **Lemma 12.** *Let $k \in \mathbb{N}$, $\mathcal{K} = (\mathcal{AP}, W, \delta, \mu, v_0)$ be a finite Kripke structure and ρ_1, ρ_1', ρ_2, ρ_2' be tracks in $\mathrm{Trk}_{\mathcal{K}}$ such that $(\mathrm{lst}(\rho_1), \mathrm{fst}(\rho_1')) \in \delta$, $(\mathrm{lst}(\rho_2), \mathrm{fst}(\rho_2')) \in \delta$, $\rho_1 \sim_k \rho_2$ and $\rho_1' \sim_k \rho_2'$. Then $\rho_1 \cdot \rho_1' \sim_k \rho_2 \cdot \rho_2'$.*

The next proposition immediately follows from Lemma 12.

▶ **Proposition 13** (Left and right extensions)**.** *Let $\mathcal{K} = (\mathcal{AP}, W, \delta, \mu, v_0)$ be a finite Kripke structure, ρ, ρ' be two tracks in $\mathrm{Trk}_{\mathcal{K}}$ such that $\rho \sim_k \rho'$, and $\overline{\rho} \in \mathrm{Trk}_{\mathcal{K}}$. If $(\mathrm{lst}(\rho), \mathrm{fst}(\overline{\rho})) \in \delta$, then $\rho \cdot \overline{\rho} \sim_k \rho' \cdot \overline{\rho}$, and if $(\mathrm{lst}(\overline{\rho}), \mathrm{fst}(\rho)) \in \delta$, then $\overline{\rho} \cdot \rho \sim_k \overline{\rho} \cdot \rho'$.*

The next theorem proves that, for any pair of tracks $\rho, \rho' \in \mathrm{Trk}_{\mathcal{K}}$, if $\rho \sim_k \rho'$, then ρ and ρ' are k-equivalent (see Definition 8). Since the set of B_k-descriptors for the tracks of a finite Kripke structure $\mathcal{K}$ is finite (or, in other words, the equivalence relation $\sim_k$ has a finite index), there always exists a finite number of B_k-descriptors that "satisfy" an $HS[A, \overline{A}, B, \overline{B}, \overline{E}]$ formula ψ with $\mathrm{Nest_B}(\psi) = k$ (this can be formally proved by a quotient construction [16]).

▶ **Theorem 14** ([16])**.** *Let $\mathcal{K}$ be a finite Kripke structure, ρ and ρ' be two tracks in $\mathrm{Trk}_{\mathcal{K}}$, $\mathcal{A}_{\mathcal{K}}$ be the abstract interval model induced by $\mathcal{K}$, and ψ be a formula of $HS[A, \overline{A}, B, \overline{B}, \overline{E}]$ with $\mathrm{Nest_B}(\psi) = k$. If $\rho \sim_k \rho'$, then $\mathcal{A}_{\mathcal{K}}, \rho \models \psi \iff \mathcal{A}_{\mathcal{K}}, \rho' \models \psi$.*

3 Clusters and descriptor element indistinguishability

A B_k-descriptor provides a finite encoding for a possibly infinite set of tracks (the tracks associated with that descriptor). Unfortunately, the representation of B_k-descriptors as trees

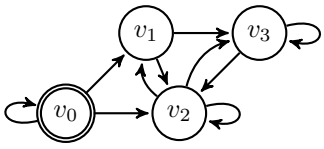


Figure 3 An example of finite Kripke structure.

labelled over descriptor elements is highly redundant. For example, given any pair of subtrees rooted in some children of the root of a descriptor, it is always the case that one of them is a subtree of the other: the two subtrees are associated with two (different) prefixes of a track and one of them is necessarily a prefix of the other. In practice, the size of the tree representation of B_k-descriptors prevents their direct use in model checking algorithms, and makes it difficult to determine the intrinsic complexity of B_k-descriptors.

In this section, we devise a more compact representation of B_k-descriptors. Each class of the k-descriptor equivalence relation is a set of k-equivalent tracks. For every such class, we select a track representative whose length is (exponentially) bounded in both the size of W (the set of states of the Kripke structure) and k. In order to set such a bound, we consider suitable ordered sequences (possibly with repetitions) of descriptor elements of a B_k-descriptor. Let us define the *descriptor sequence* for a track as the ordered sequence of descriptor elements associated with its prefixes. In a descriptor sequence, descriptor elements can obviously be repeated: we devise a criterion to avoid such repetitions whenever they cannot be distinguished by any $HS[A, \overline{A}, B, \overline{B}, \overline{E}]$ formula of B-nesting depth up to k.

▶ **Definition 15.** Let $\rho = v_0 v_1 \dots v_n$ be a track of a finite Kripke structure. The descriptor sequence ρ_{ds} for ρ is $d_0 \dots d_{n-1}$, where $d_i = \rho_{ds}(i) = (v_0, \text{intstates}(v_0 \dots v_{i+1}), v_{i+1})$, for $i \in \{0, \dots, n-1\}$. We denote the set of descriptor elements occurring in ρ_{ds} by $DElm(\rho_{ds})$.

For example, let us consider the finite Kripke structure of Figure 3 and the track $\rho = v_0 v_0 v_0 v_1 v_2 v_1 v_2 v_3 v_3 v_2 v_2$. The descriptor sequence for ρ is:

$$\rho_{ds} = (v_0, \emptyset, v_0)\boxed{(v_0, \{v_0\}, v_0)}(v_0, \{v_0\}, v_1)(v_0, \{v_0, v_1\}, v_2)$$
$$\boxed{(v_0, \Gamma, v_1)(v_0, \Gamma, v_2)}(v_0, \Gamma, v_3)\boxed{(v_0, \Delta, v_3)(v_0, \Delta, v_2)(v_0, \Delta, v_2)}, \quad (*)$$

where $\Gamma = \{v_0, v_1, v_2\}$, $\Delta = \{v_0, v_1, v_2, v_3\}$. $DElm(\rho_{ds})$ is the set $\{(v_0, \emptyset, v_0), (v_0, \{v_0\}, v_0), (v_0, \{v_0\}, v_1), (v_0, \{v_0, v_1\}, v_2), (v_0, \Gamma, v_1), (v_0, \Gamma, v_2), (v_0, \Gamma, v_3), (v_0, \Delta, v_2), (v_0, \Delta, v_3)\}$.

To express the relationships between descriptor elements occurring in a descriptor sequence, we introduce a binary relation, $\mathrm{R_t}$. Intuitively, given two descriptor elements d' and d'' of a descriptor sequence, the relation $d' \,\mathrm{R_t}\, d''$ holds if d' and d'' are the descriptor elements of two tracks ρ' and ρ'', respectively, and ρ' is a prefix of ρ''.

▶ **Definition 16.** Let ρ_{ds} be the descriptor sequence for a track ρ and let $d' = (v_{in}, S', v'_{fin})$ and $d'' = (v_{in}, S'', v''_{fin})$ be two descriptor elements in ρ_{ds}. Then, $d' \,\mathrm{R_t}\, d''$ iff $S' \cup \{v'_{fin}\} \subseteq S''$.

The relation $\mathrm{R_t}$ is transitive: for all descriptor elements d', d'', d''', if $d' \,\mathrm{R_t}\, d''$ and $d'' \,\mathrm{R_t}\, d'''$, then $S' \cup \{v'_{fin}\} \subseteq S''$ and $S'' \cup \{v''_{fin}\} \subseteq S'''$; it follows that $S' \cup \{v'_{fin}\} \subseteq S'''$, and thus $d' \,\mathrm{R_t}\, d'''$. $\mathrm{R_t}$ is neither an equivalence relation nor a quasiorder, since $\mathrm{R_t}$ is neither reflexive (e.g., $(v_0, \{v_0\}, v_1) \not\mathrm{R_t} (v_0, \{v_0\}, v_1)$), nor symmetric (e.g., $(v_0, \{v_0\}, v_1)\,\mathrm{R_t}(v_0, \{v_0, v_1\}, v_1)$ and $(v_0, \{v_0, v_1\}, v_1) \not\mathrm{R_t} (v_0, \{v_0\}, v_1)$), nor antisymmetric (e.g., $(v_0, \{v_1, v_2\}, v_1)\,\mathrm{R_t}(v_0, \{v_1, v_2\}, v_2)$ and $(v_0, \{v_1, v_2\}, v_2)\,\mathrm{R_t}\ (v_0, \{v_1, v_2\}, v_1)$, but the two elements are distinct).

It can be easily shown that $\mathrm{R_t}$ pairs descriptor elements of increasing prefixes of a track.

▶ **Proposition 17.** *Let ρ_{ds} be the descriptor sequence for the track $\rho = v_0 v_1 \cdots v_n$. Then, $\rho_{ds}(i)\,\mathrm{R_t}\,\rho_{ds}(j)$ for all $0 \leq i < j < n$.*

We now introduce a distinction between two types of descriptor elements.

▶ **Definition 18.** A descriptor element (v_{in}, S, v_{fin}) is a Type-1 descriptor element if $v_{fin} \notin S$, while it is a Type-2 descriptor element if $v_{fin} \in S$.

It can be easily checked that a descriptor element $d = (v_{in}, S, v_{fin})$ is Type-1 if and only if $\mathrm{R_t}$ is not reflexive in d: (i) if $d \not\mathrel{\mathrm{R_t}} d$, then $S \cup \{v_{fin}\} \not\subseteq S$, and thus $v_{fin} \notin S$, and (ii) if $v_{fin} \notin S$, then $d \not\mathrel{\mathrm{R_t}} d$. It follows that a Type-1 descriptor element cannot occur more than once in a descriptor sequence. On the other hand, Type-2 descriptor elements may occur multiple times in a descriptor sequence, and if a descriptor element occurs more than once, then it is necessarily of Type-2.

▶ **Proposition 19.** *If both $d'\,\mathrm{R_t}\,d''$ and $d''\,\mathrm{R_t}\,d'$ for $d' = (v_{in}, S', v'_{fin})$ and $d'' = (v_{in}, S'', v''_{fin})$ then $v'_{fin} \in S'$, $v''_{fin} \in S''$ and $S' = S''$; thus both d' and d'' are Type-2 descriptor elements.*

We are now ready to give a general characterization of the descriptor sequence ρ_{ds} for a track ρ: ρ_{ds} is composed of some (maximal) subsequences, consisting of occurrences of Type-2 descriptor elements on which $\mathrm{R_t}$ is symmetric, separated by occurrences of Type-1 descriptor elements. This can be formalized by means of the notion of cluster.

▶ **Definition 20.** A cluster $\mathcal{C}$ of (Type-2) descriptor elements is a maximal set of descriptor elements $\{d_1, \ldots, d_s\} \subseteq DElm(\rho_{ds})$ such that $d_i\,\mathrm{R_t}\,d_j$ and $d_j\,\mathrm{R_t}\,d_i$ for all $i, j \in \{1, \ldots, s\}$.

Thanks to maximality, clusters are pairwise disjoint: if $\mathcal{C}$ and $\mathcal{C}'$ are distinct clusters, $d \in \mathcal{C}$ and $d' \in \mathcal{C}'$, either $d\,\mathrm{R_t}\,d'$ and $d' \not\mathrel{\mathrm{R_t}} d$, or $d'\,\mathrm{R_t}\,d$ and $d \not\mathrel{\mathrm{R_t}} d'$.

It can be easily checked that the descriptor elements of a cluster $\mathcal{C}$ are contiguous in ρ_{ds} (in other words, they form a subsequence of ρ_{ds}), that is, occurrences of descriptor elements of $\mathcal{C}$ are never shuffled with occurrences of descriptor elements not belonging to $\mathcal{C}$.

▶ **Definition 21.** Let ρ_{ds} be a descriptor sequence and $\mathcal{C}$ be one of its clusters. The subsequence of ρ_{ds} associated with $\mathcal{C}$ is the subsequence $\rho_{ds}(i, j)$, with $i \leq j < |\rho_{ds}|$, including all and only the occurrences of the descriptor elements in $\mathcal{C}$.

Notice that two subsequences associated with two distinct clusters $\mathcal{C}$ and $\mathcal{C}'$ in a descriptor sequence must be separated by at least one occurrence of a Type-1 descriptor element. For example, with reference to the descriptor sequence (*) for $\rho = v_0 v_0 v_0 v_1 v_2 v_1 v_2 v_3 v_3 v_2 v_2$ of the Kripke structure in Figure 3, the subsequences associated with clusters are enclosed in boxes.

While $\mathrm{R_t}$ allows us to order any pair of Type-1 descriptor elements, as well as any Type-1 descriptor element with respect to a Type-2 descriptor element, it does not give any means to order Type-2 descriptor elements belonging to the same cluster. This, together with the fact that Type-2 elements may have multiple occurrences in a descriptor sequence, implies that we need to somehow limit the number of occurrences of Type-2 elements in order to give a bound on the length of track representatives of B_k-descriptors.

To this end, we introduce an equivalence relation that allows us to put together indistinguishable occurrences of the same descriptor element in a descriptor sequence, that is, to detect those occurrences which are associated with prefixes of the track with the same B_k-descriptor. The idea is that a track representative for a B_k-descriptor should not include indistinguishable occurrences of the same descriptor element.

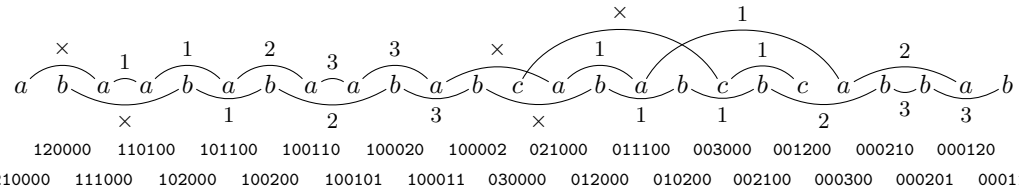


Figure 4 The track $\rho = v_0v_1v_2v_3v_3v_2v_3v_3v_2v_3v_2v_3v_3v_2v_3v_2v_1v_3v_2v_3v_2v_1v_2v_1v_3v_2v_2v_3v_2$ of the finite Kripke structure depicted in Figure 3 generates the descriptor sequence $\rho_{ds} = (v_0, \emptyset, v_1)(v_0, \{v_1\}, v_2)(v_0, \{v_1, v_2\}, v_3)abaababaababcababcbcabbab$, where a, b, and c stand for $(v_0, \{v_1, v_2, v_3\}, v_3)$, $(v_0, \{v_1, v_2, v_3\}, v_2)$, and $(v_0, \{v_1, v_2, v_3\}, v_1)$, respectively. Here we show the subsequence $\rho_{ds}(3, |\rho_{ds}| - 1)$ associated with the cluster $\mathcal{C} = \{a, b, c\}$. Pairs of k-indistinguishable consecutive occurrences of descriptor elements are connected by a rounded edge labelled by k. Edges labelled by $\times$ link occurrences which are not 1-indistinguishable. The values of all missing edges can be derived from the properties established by Proposition 24 and 26. At the bottom of the figure, for each position, we give the associated configurations: $c(3) = (2, 1, 0, 0, 0, 0)$, $c(4) = (1, 2, 0, 0, 0, 0)$, and so forth.

▶ **Definition 22.** Let ρ_{ds} be a descriptor sequence and $k \geq 1$. We say that two occurrences $\rho_{ds}(i)$ and $\rho_{ds}(j)$, with $0 \leq i < j < |\rho_{ds}|$, *of the same descriptor element* d are k-indistinguishable if (and only if):

- (for $k = 1$) $DElm(\rho_{ds}(0, i-1)) = DElm(\rho_{ds}(0, j-1))$;
- (for $k \geq 2$) for all $i \leq \ell \leq j-1$, there exists $0 \leq \ell' \leq i-1$ such that $\rho_{ds}(\ell)$ and $\rho_{ds}(\ell')$ are $(k-1)$-indistinguishable.

From Definition 22, it follows that two indistinguishable occurrences $\rho_{ds}(i)$ and $\rho_{ds}(j)$ of the same descriptor element necessarily belong to the same subsequence of ρ_{ds} associated with a cluster. In general, it is always the case that $DElm(\rho_{ds}(0, i-1)) \subseteq DElm(\rho_{ds}(0, j-1))$ for $i < j$; 1-indistinguishability also guarantees $DElm(\rho_{ds}(0, i-1)) = DElm(\rho_{ds}(0, j-1))$. From this, it easily follows that the two first occurrences of a descriptor element are not 1-indistinguishable.

Proposition 23 and 24 state some basic properties of the k-indistinguishability relation.

▶ **Proposition 23.** *Let $k \geq 2$ and $\rho_{ds}(i)$ and $\rho_{ds}(j)$, with $0 \leq i < j < |\rho_{ds}|$, be two k-indistinguishable occurrences of the same descriptor element in a descriptor sequence ρ_{ds}. Then, $\rho_{ds}(i)$ and $\rho_{ds}(j)$ are also $(k-1)$-indistinguishable.*

▶ **Proposition 24.** *Let $k \geq 1$ and $\rho_{ds}(i)$ and $\rho_{ds}(m)$, with $0 \leq i < m < |\rho_{ds}|$, be two k-indistinguishable occurrences of the same descriptor element in a descriptor sequence ρ_{ds}. If $\rho_{ds}(j) = \rho_{ds}(m)$, for some $i < j < m$, then $\rho_{ds}(j)$ and $\rho_{ds}(m)$ are k-indistinguishable.*

In Figure 4, we give some examples of k-indistinguishability relations, for $k \in \{1, 2, 3\}$, for a track of the finite Kripke structure depicted in Figure 3.

The next theorem establishes a fundamental connection between k-indistinguishability of descriptor elements and k-descriptor equivalence of tracks.

▶ **Theorem 25.** *Let ρ_{ds} be the descriptor sequence for a track ρ. Two occurrences $\rho_{ds}(i)$ and $\rho_{ds}(j)$, with $0 \leq i < j < |\rho_{ds}|$, of the same descriptor element are k-indistinguishable if and only if $\rho(0, i+1) \sim_k \rho(0, j+1)$.*

Notice that k-indistinguishability between occurrences of descriptor elements is defined *only for pairs of prefixes of the same track*, while the relation of k-descriptor equivalence can be applied to pairs of any tracks of a Kripke structure.

The next proposition easily follows from Theorem 25.

▶ **Proposition 26.** *Let $\rho_{ds}(i)$, $\rho_{ds}(j)$, and $\rho_{ds}(m)$, with $0 \leq i < j < m < |\rho_{ds}|$, be three occurrences of the same descriptor element. If both the pair $\rho_{ds}(i)$ and $\rho_{ds}(j)$ and the pair $\rho_{ds}(j)$ and $\rho_{ds}(m)$ are k-indistinguishable, for some $k \geq 1$, then $\rho_{ds}(i)$ and $\rho_{ds}(m)$ are k-indistinguishable, as well.*

4 A model checking procedure based on track representatives

In this section, we will exploit the k-indistinguishability relation between descriptor elements in a descriptor sequence ρ_{ds} for a track ρ to possibly replace ρ by a k-descriptor equivalent, *shorter* track ρ' of bounded length. This allows us to find, for each B_k-descriptor $\mathcal{D}_{B_k}$ (witnessed by a track of the considered finite Kripke structure $\mathcal{K}$), a *track representative* $\tilde{\rho}$ in $\mathcal{K}$ such that (i) $\mathcal{D}_{B_k}$ is the B_k-descriptor for $\tilde{\rho}$ and (ii) the length of $\tilde{\rho}$ is bounded. Thanks to property (ii), we can check all the track representatives of a finite Kripke structure by simply visiting its unravelling up to a bounded depth.

The notion of track representative can be explained as follows. Let ρ_{ds} be the descriptor sequence for a track ρ. If there are two occurrences of the same descriptor element $\rho_{ds}(i)$ and $\rho_{ds}(j)$, with $i < j$, which are k-indistinguishable (we let $\rho = \rho(0, j+1) \cdot \overline{\rho}$, with $\overline{\rho} = \rho(j+2, |\rho|-1)$), then we can replace ρ by the k-descriptor equivalent, shorter track $\rho(0, i+1) \cdot \overline{\rho}$: by Theorem 25, $\rho(0, i+1)$ and $\rho(0, j+1)$ have the same B_k-descriptor and thus, by Proposition 13, $\rho = \rho(0, j+1) \cdot \overline{\rho}$ and $\rho(0, i+1) \cdot \overline{\rho}$ have the same B_k-descriptor. Moreover, since $\rho_{ds}(i)$ and $\rho_{ds}(j)$ are occurrences of the same descriptor element, $\rho(i+1) = \rho(j+1)$ and so the track $\rho(0, i+1) \cdot \overline{\rho}$ is witnessed in the finite Kripke structure. By iteratively applying such a contraction method, we can find a track ρ' which is k-descriptor equivalent to ρ, whose descriptor sequence is devoid of k-indistinguishable occurrences of descriptor elements. *A track representative* is a track that fulfils this property.

We now show how to give a bound to the length of track representatives. We start by stating some technical properties. The next proposition provides a bound to the distance within which we observe a repeated occurrence of some descriptor element in the descriptor sequence for a track. We preliminarily observe that, for any track ρ, $|DElm(\rho_{ds})| \leq |W|^2 + 1$, where W is the set of states of the finite Kripke structure. Indeed, in the descriptor sequence, the sets of internal states of prefixes of ρ increase monotonically with respect to the "$\subseteq$" relation. As a consequence, at most $|W|$ distinct sets may occur, excluding $\emptyset$ which can occur only in the first descriptor element. Moreover, these sets can be paired with all possible final states which are at most $|W|$.

▶ **Proposition 27.** *For each track ρ of $\mathcal{K}$, associated with a descriptor element d, there exists a track ρ' of $\mathcal{K}$, associated with the same descriptor element d, such that $|\rho'| \leq 2 + |W|^2$.*

Proposition 27 will be used in the unravelling Algorithm 1 as a termination criterion (referred to as *0-termination criterion*) for unravelling a finite Kripke structure when it is not necessary to observe multiple occurrences of the same descriptor element: *to get a track representative for all descriptor elements, witnessed in a finite Kripke structure with set of states W and initial state v, we can avoid considering tracks longer than $2 + |W|^2$, while exploring the unravelling of the Kripke structure from v.*

Let us now consider the problem of establishing a bound for tracks devoid of pairs of k-indistinguishable occurrences of descriptor elements. We first notice that, in a descriptor sequence ρ_{ds} for a track ρ, there are at most $|W|$ occurrences of Type-1 descriptor elements. On the contrary, Type-2 descriptor elements can occur multiple times and thus, to bound the length of ρ_{ds}, one has to constrain the *number* and the *length* of the subsequences of ρ_{ds}

associated with clusters. As for their number, it suffices to observe that they are separated by Type-1 descriptor elements, and hence at most $|W|$ of them, related to distinct clusters, can occur in a descriptor sequence.

As for their length, we can proceed as follows. First, for any cluster $\mathcal{C}$, it holds that $|\mathcal{C}| \leq |W|$ as all (Type-2) descriptor elements of $\mathcal{C}$ share the same set S of internal states and their final states v_{fin} must belong to S. In the following, we consider the (maximal) subsequence $\rho_{ds}(u, v)$ of ρ_{ds} associated with a specific cluster $\mathcal{C}$, for some $0 \leq u \leq v \leq |\rho_{ds}|-1$, and when we mention an index i, we implicitly assume that $u \leq i \leq v$, that is, i refers to a position in the subsequence. We sequentially scan such a subsequence suitably recording the multiplicity of occurrences of descriptor elements into an auxiliary structure. To detect indistinguishable occurrences of descriptor elements up to indistinguishability $s \geq 1$, we use $s + 3$ arrays $Q_{-2}()$, $Q_{-1}()$, $Q_0()$, $Q_1()$, ..., $Q_s()$. Array elements are sets of descriptor elements of $\mathcal{C}$. Given an index i, the sets at position i, $Q_{-2}(i)$, $Q_{-1}(i)$, $Q_0(i)$, $Q_1(i)$, ..., $Q_s(i)$, store information about indistinguishabilty for multiple occurrences of descriptor elements in the subsequence up to position $i > u$. To exemplify, if the scan function finds an occurrence of the descriptor element $d \in \mathcal{C}$ at position i, that is, $\rho_{ds}(i) = d$, we have that:

1. $Q_{-2}(i)$ contains all descriptor elements of $\mathcal{C}$ which have never occurred in $\rho_{ds}(u, i)$;
2. $d \in Q_{-1}(i)$ if d has never occurred in $\rho_{ds}(u, i-1)$ and $\rho_{ds}(i) = d$, that is, $\rho_{ds}(i)$ is the first occurrence of d in $\rho_{ds}(u, i)$;
3. $d \in Q_0(i)$ if d occurs at least twice in $\rho_{ds}(u, i)$ and the occurrence $\rho_{ds}(i)$ of d is *not* 1-indistinguishable from the last occurrence of d in $\rho_{ds}(u, i-1)$;
4. $d \in Q_t(i)$ (for some $t \geq 1$) if the occurrence $\rho_{ds}(i)$ of d is t-indistinguishable, but *not* $(t+1)$-indistinguishable, from the last occurrence of d in $\rho_{ds}(u, i-1)$.

In particular, at position u (the first of the subsequence), $Q_{-1}(u)$ contains only the descriptor element $d = \rho_{ds}(u)$, $Q_{-2}(u)$ is the set $\mathcal{C} \setminus \{d\}$, and $Q_0(u)$, $Q_1(u)$, ... are empty sets.

In general, arrays $Q_{-2}()$, $Q_{-1}()$, $Q_0()$, $Q_1()$, ..., $Q_s()$ satisfy the following constraints: for all i, $\bigcup_{m=-2}^{s} Q_m(i) = \mathcal{C}$ and, for all i and all $m \neq m'$, $Q_m(i) \cap Q_{m'}(i) = \emptyset$.

Intuitively, at every position i, $Q_{-2}(i)$, $Q_{-1}(i), Q_0(i), Q_1(i)$, ..., $Q_s(i)$ describe a *state* of the scanning process of the subsequence. The change of the state produced by the transition from position $i-1$ to position i while scanning the sequence is formally defined by the function f, reported in Figure 5, which maps the descriptor sequence ρ_{ds} and a position i to the tuple of sets $\big(Q_{-2}(i), Q_{-1}(i), Q_0(i), Q_1(i), \ldots, Q_s(i)\big)$.

Notice that, whenever a descriptor element $\rho_{ds}(i) = d$ is such that $d \in Q_z(i-1)$ and $d \in Q_{z'}(i)$, with $z < z'$ (cases (a), (b) and (d) of the definition of f), all $Q_{z''}(i)$, with $z'' > z'$, are empty sets and, for all $z'' \geq z'$, all elements in $Q_{z''}(i-1)$ belong to $Q_{z'}(i)$. Consider, for instance, the following scenario: in a subsequence of ρ_{ds}, associated with some cluster $\mathcal{C}$, $\rho_{ds}(h) = \rho_{ds}(i) = d \in \mathcal{C}$ and $\rho_{ds}(h') = \rho_{ds}(i') = d' \in \mathcal{C}$, for some $h < h' < i < i'$ and $d \neq d'$, and there are not other occurrences of d and d' in $\rho_{ds}(h, i')$. If $\rho_{ds}(h)$ and $\rho_{ds}(i)$ are exactly z'-indistinguishable, by definition of the indistinguishability relation, $\rho_{ds}(h')$ and $\rho_{ds}(i')$ can be no more than $(z'+1)$-indistinguishable. Thus, if d' is in $Q_{z''}(i-1)$, for some $z'' > z'$, we can safely "downgrade" it to $Q_{z'}(i)$, because we know that, when we meet the next occurrence of d' ($\rho_{ds}(i')$), $\rho_{ds}(h')$ and $\rho_{ds}(i')$ will be no more than $(z'+1)$-indistinguishable.

In the following, we will make use of an abstract characterisation of the state of the arrays at a given position i, as determined by the scan function f, called *configuration*, that only considers the cardinality of the sets of arrays. Theorem 29 states that, when a descriptor subsequence is scanned, configurations never repeat since the sequence of configurations is

$f(\rho_{ds}, u) = \big(\mathcal{C} \setminus \{d\}, \{d\}, \emptyset, \cdots, \emptyset\big)$ with $\rho_{ds}(u) = d$;

For all $i > u$: $f(\rho_{ds}, i) = \big(Q_{-2}(i), Q_{-1}(i), Q_0(i), \ldots, Q_s(i)\big) =$

$$\begin{cases} \big(Q_{-2}(i-1) \setminus \{d\}, \{d\} \cup \bigcup_{m=-1}^{s} Q_m(i-1), \emptyset, \ldots, \emptyset\big) \text{ if } \rho_{ds}(i) \text{ is the first occurrence of } d \text{ in } \rho_{ds}(u,i); \textbf{(a)} \\ \big(Q_{-2}(i-1), Q_{-1}(i-1) \setminus \{d\}, \{d\} \cup \bigcup_{m=0}^{s} Q_m(i-1), \emptyset, \ldots, \emptyset\big) \text{ if } \rho_{ds}(i) = d,\ d \in Q_{-1}(i-1), \text{ and } \rho_{ds}(i) \text{ is at least the second occurrence of } d \text{ in } \rho_{ds}(u,i) \text{ and it is } \textit{not} \text{ 1-indistinguishable from the immediately preceding occurrence of } d; \textbf{(b)} \\ \big(Q_{-2}(i-1), Q_{-1}(i-1), \{d\} \cup Q_0(i-1), Q_1(i-1) \setminus \{d\}, \ldots, Q_s(i-1) \setminus \{d\}\big) \text{ if } \rho_{ds}(i) = d,\ d \in \bigcup_{m=0}^{s} Q_m(i-1), \text{ and } \rho_{ds}(i) \text{ is at least the second occurrence of } d \text{ in } \rho_{ds}(u,i) \text{ and it is } \textit{not} \text{ 1-indistinguishable from the immediately preceding occurrence of } d; \textbf{(c)} \\ \big(Q_{-2}(i-1) \setminus \{d\}, \ldots, Q_{t-1}(i-1) \setminus \{d\}, \{d\} \cup \bigcup_{m=t}^{s} Q_m(i-1), \emptyset, \ldots, \emptyset\big) \text{ if } \rho_{ds}(i) = d,\ \rho_{ds}(i) \text{ is } t\text{-indistinguishable (for some } t \geq 1), \text{ but } \textit{not } (t+1)\text{-indistinguishable, to the immediately preceding occurrence of } d, \text{ and } d \in \bigcup_{m=-2}^{t-1} Q_m(i-1); \textbf{(d)} \\ \big(Q_{-2}(i-1), \cdots, Q_{t-1}(i-1), \{d\} \cup Q_t(i-1), Q_{t+1}(i-1) \setminus \{d\}, \ldots, Q_s(i-1) \setminus \{d\}\big) \text{ if } \rho_{ds}(i) = d,\ \rho_{ds}(i) \text{ is } t\text{-indistinguishable (for some } t \geq 1), \text{ but } \textit{not } (t+1)\text{-indistinguishable, to the immediately preceding occurrence of } d, \text{ and } d \in \bigcup_{m=t}^{s} Q_m(i-1). \textbf{(e)} \end{cases}$$

Figure 5 Definition of the scan function f.

strictly decreasing according to the lexicographical order $>_{lex}$. This property will allow us to establish the desired bound to the length of track representatives.

▶ **Definition 28.** Let ρ_{ds} be the descriptor sequence for a track ρ and i be a position in the subsequence of ρ_{ds} associated with a given cluster. The *configuration at position* i, written $c(i)$, is the tuple $c(i) = (|Q_{-2}(i)|, |Q_{-1}(i)|, |Q_0(i)|, |Q_1(i)|, \cdots, |Q_s(i)|)$, where $f(\rho_{ds}, i) = (Q_{-2}(i), Q_{-1}(i), Q_0(i), Q_1(i), \cdots, Q_s(i))$.

An example of a configuration sequence is given in Figure 4.

▶ **Theorem 29.** *Let ρ_{ds} be the descriptor sequence for a track ρ and $\rho_{ds}(u,v)$, for some $u < v$, be the subsequence associated with a cluster $\mathcal{C}$. For all $u < i \leq v$, if $\rho_{ds}(i) = d$, then it holds that $d \in Q_t(i-1)$, $d \in Q_{t+1}(i)$, and $c(i-1) >_{lex} c(i)$, for some $t \in \{-2, -1\} \cup \mathbb{N}$.*

We show now how to select all and only those tracks which do not feature any pair of k-indistinguishable occurrences of descriptor elements. To this end, we make use of a scan function f which exploits $k+3$ arrays (the value $k+3$ derives from the k of descriptor element indistinguishability, plus the three arrays $Q_{-2}()$, $Q_{-1}()$, $Q_0()$). Theorem 29 guarantees that, while scanning a subsequence, configurations are never repeated. This allows us to set an upper bound to the length of a track such that, whenever exceeded, the descriptor sequence for the track features at least a pair of k-indistinguishable occurrences of a descriptor element. The bound is essentially given by the number of possible configurations for $k+3$ arrays.

By an easy combinatorial argument, we can prove the following proposition.

▶ **Proposition 30.** *For all $n, t \in \mathbb{N}^+$, the number of distinct t-tuples of natural numbers whose sum equals n is $\varepsilon(n,t) = \binom{n+t-1}{n} = \binom{n+t-1}{t-1}$.*

Proposition 30 provides two upper bounds for $\varepsilon(n,t)$: $\varepsilon(n,t) \leq (n+1)^{t-1}$ and $\varepsilon(n,t) \leq t^n$.

Since a configuration $c(i)$ of a cluster $\mathcal{C}$ is a $(k+3)$-tuple whose elements add up to $|\mathcal{C}|$, Proposition 30 allows us to conclude that there are at most $\varepsilon(|\mathcal{C}|, k+3) = \binom{|\mathcal{C}|+k+2}{k+2}$ distinct configurations of size $(k+3)$, whose integers add up to $|\mathcal{C}|$. Moreover, since configurations never repeat while scanning a subsequence associated with a cluster $\mathcal{C}$, $\varepsilon(|\mathcal{C}|, k+3)$ is an upper bound to the length of such a subsequence.

Now, for any track ρ, ρ_{ds} has at most $|W|$ subsequences associated with distinct clusters $\mathcal{C}_1, \mathcal{C}_2, \ldots$, and thus if the following upper bound to the length of ρ is exceeded, then there is at least one pair of k-indistinguishable occurrences of a descriptor element in ρ_{ds}: $|\rho| \leq 1 + (|\mathcal{C}_1|+1)^{k+2} + (|\mathcal{C}_2|+1)^{k+2} + \cdots + (|\mathcal{C}_s|+1)^{k+2} + |W|$, where $s \leq |W|$ and the last addend is to count occurrences of Type-1 descriptor elements. Since clusters are disjoint and their union is a subset of $DElm(\rho_{ds})$, and $|DElm(\rho_{ds})| \leq 1 + |W|^2$, we get two upper bounds:

$$|\rho| \leq 1 + (|\mathcal{C}_1| + |\mathcal{C}_2| + \cdots + |\mathcal{C}_s| + |W|)^{k+2} + |W| \leq 1 + (|DElm(\rho_{ds})| + |W|)^{k+2} + |W| \leq 1 + (1 + |W|^2 + |W|)^{k+2} + |W| \leq 1 + (1+|W|)^{2k+4} + |W|,$$

and, analogously,

$$|\rho| \leq 1 + (k+3)^{|\mathcal{C}_1|} + (k+3)^{|\mathcal{C}_2|} + \cdots + (k+3)^{|\mathcal{C}_s|} + |W| \leq 1 + (k+3)^{|\mathcal{C}_1|+|\mathcal{C}_2|+\cdots+|\mathcal{C}_s|} + |W| \leq 1 + (k+3)^{|DElm(\rho_{ds})|} + |W| \leq 1 + (k+3)^{|W|^2+1} + |W|.$$

The upper bound for $|\rho|$ is then the least of the two given upper bounds:

$$\tau(|W|, k) = \min\left\{1 + (1+|W|)^{2k+4} + |W|, 1 + (k+3)^{|W|^2+1} + |W|\right\}.$$

▶ **Theorem 31.** *Let $\mathcal{K}$ be a finite Kripke structure and ρ be a track in $Trk_{\mathcal{K}}$. If $|\rho| > \tau(|W|, k)$, there exists another track in $\mathrm{Trk}_{\mathcal{K}}$, whose length is less than or equal to $\tau(|W|, k)$, which has the same B_k-descriptor as ρ.*

Theorem 31 allows us to define a termination criterion to bound the depth of the unravelling of a finite Kripke structure ($(k \geq 1)$-*termination criterion*), while searching for track representatives for witnessed B_k-descriptors: *for any $k \geq 1$, to get a track representative for every B_k-descriptor with initial state v and witnessed in a finite Kripke structure with set of states W, we can avoid taking into consideration tracks longer than $\tau(|W|, k)$ while exploring the unravelling of the structure from v.*

Algorithm 1 (the *unravelling algorithm*) explores the unravelling of the input Kripke structure $\mathcal{K}$ to find the track representatives for all witnessed B_k-descriptors. The upper bound $\tau(|W|, k)$ on the maximum depth of the unravelling ensures the termination of the algorithm, which never returns a track ρ if there exist k-indistinguishable occurrences of a descriptor element in ρ_{ds}.

The next theorem proves soundness and completeness of Algorithm 1.

▶ **Theorem 32.** *Let $\mathcal{K}$ be a finite Kripke structure, v be a state in W, and $k \in \mathbb{N}$. For every track ρ of $\mathcal{K}$, with $\mathrm{fst}(\rho) = v$ and $|\rho| \geq 2$, the unravelling algorithm returns a track ρ' of $\mathcal{K}$, with $\mathrm{fst}(\rho') = v$, such that ρ and ρ' have the same B_k-descriptor and $|\rho'| \leq \tau(|W|, k)$.*

As an example, $\rho' = v_0v_1v_2v_3v_3v_2v_3v_3v_2v_3v_2v_3v_2v_1v_3v_2v_3v_2v_1v_2v_1v_3v_2$ is returned by Algorithm 1 in place of the track ρ of Figure 4; it can be checked that ρ'_{ds} does not contain any pair of 3-indistinguishable occurrences of a descriptor element and that ρ and ρ' have the same B_3-descriptor.

Algorithm 1 `Unrav`$(\mathcal{K}, v, k, \text{direction})$

if direction = FORW **then**
  Unravel $\mathcal{K}$ starting from v according to $\ll$ ▷ "$\ll$" is an arbitrary order of the nodes of $\mathcal{K}$
  For every new node *of the unravelling* met during the visit, return the track ρ from v to the current node only if:
  if $k = 0$ **then**
    Apply the 0-termination criterion
  else
    if The last descriptor element d of (the descriptor sequence of) the current track ρ is k-indistinguishable from a previous occurrence of d **then**
      do *not* return ρ and backtrack to $\rho(0, |\rho| - 2) \cdot \overline{v}$, where $\overline{v}$ is the minimum state (w.r.t. $\ll$) greater than $\rho(|\rho| - 1)$ such that $(\rho(|\rho| - 2), \overline{v})$ is an edge of $\mathcal{K}$.
else if direction = BACKW **then**
  Unravel $\overline{\mathcal{K}}$ starting from v according to $\ll$ ▷ $\overline{\mathcal{K}}$ is $\mathcal{K}$ with transposed edges
  For every new node *of the unravelling* met during the visit, consider the track ρ *from the current node to* v, and recalculate descriptor elements indistinguishability from scratch (left to right); return the track only if:
  if $k = 0$ **then**
    Apply the 0-termination criterion
  else
    if There exist two k-indistinguishable occurrences of a descriptor element d in (the descriptor sequence of) the current track ρ **then**
      do *not* return ρ
  Do not visit tracks of length greater than $\tau(|W|, k)$

Algorithm 2 `ModCheck`$(\mathcal{K}, \psi)$

$k \leftarrow \text{Nest}_{\text{B}}(\psi)$
$u \leftarrow New$ (`Unrav`$(\mathcal{K}, w_0, k, \text{FORW})$)
while u.`hasMoreTracks()` **do**
  $\tilde{\rho} \leftarrow u$.`getNextTrack()`
  if `Check`$(\mathcal{K}, k, \psi, \tilde{\rho}) = 0$ **then**
    return 0: "$\mathcal{K}, \tilde{\rho} \not\models \psi$"
return 1: "$\mathcal{K} \models \psi$"

In the *forward mode* of Algorithm 1 (used to deal with $\langle \text{A} \rangle$ and $\langle \overline{\text{B}} \rangle$ modalities), the direction of track exploration and that of indistinguishability checking are the same, so we can stop extending a track as soon as the first pair of k-indistinguishable occurrences of a descriptor element is found in the descriptor sequence, suggesting an easy termination criterion for stopping the unravelling of tracks. In the *backward mode* (exploited in the case of $\langle \overline{\text{A}} \rangle$ and $\langle \overline{\text{E}} \rangle$ modalities), such a straightforward criterion cannot be adopted, because tracks are explored right to left (the opposite direction with respect to the edges of the Kripke structure), while the indistinguishability relation over descriptor elements is computed left to right. In general, changing the prefix of a considered track requires recomputing from scratch the descriptor sequence and the indistinguishability relation over descriptor elements. In particular, k-indistinguishable occurrences of descriptor elements can be detected in the middle of a subsequence, and not necessarily at the end.

Building on Algorithm 1 we can easily define the model checking procedure `ModCheck`$(\mathcal{K}, \psi)$ (Algorithm 2). `ModCheck`$(\mathcal{K}, \psi)$ exploits the procedure `Check`$(\mathcal{K}, k, \psi, \tilde{\rho})$ (Algorithm 3) which checks a formula ψ of B-nesting depth k against a track $\tilde{\rho}$ of the Kripke structure $\mathcal{K}$. `Check`$(\mathcal{K}, k, \psi, \tilde{\rho})$ basically calls itself recursively on the subformulas of ψ, and uses the unravelling Algorithm 1 to deal with $\langle \text{A} \rangle$, $\langle \overline{\text{A}} \rangle$, $\langle \overline{\text{B}} \rangle$, and $\langle \overline{\text{E}} \rangle$ modalities.

Algorithm 3 $\texttt{Check}(\mathcal{K}, k, \psi, \tilde{\rho})$

```
1: if ψ = ⊤ then
2:   return 1
3: else if ψ = ⊥ then
4:   return 0
5: else if ψ = p ∈ AP then
6:   if p ∈ ⋂_{s∈states(ρ̃)} μ(s) then
7:     return 1 else return 0
8: else if ψ = ¬φ then
9:   return 1 − Check(K, k, φ, ρ̃)
10: else if ψ = φ1 ∧ φ2 then
11:   if Check(K, k, φ1, ρ̃) = 0 then
12:     return 0
13:   else
14:     return Check(K, k, φ2, ρ̃)
15: else if ψ = ⟨A⟩ φ then
16:   u ← New(Unrav(K, lst(ρ̃), k, FORW))
17:   while u.hasMoreTracks() do
18:     ρ ← u.getNextTrack()
19:     if Check(K, k, φ, ρ) = 1 then
20:       return 1
21:   return 0
22: else if ψ = ⟨Ā⟩ φ then
23:   u ← New(Unrav(K, fst(ρ̃), k, BACKW))
24:   while u.hasMoreTracks() do
25:     ρ ← u.getNextTrack()
26:     if Check(K, k, φ, ρ) = 1 then
27:       return 1
28:   return 0
29: else if ψ = ⟨B⟩ φ then
30:   for each ρ̄ prefix of ρ̃ do
31:     if Check(K, k − 1, φ, ρ̄) = 1 then
32:       return 1
33:   return 0
34: else if ψ = ⟨B̄⟩ φ then
35:   for each v ∈ W s.t. (lst(ρ̃), v) ∈ δ do
36:     if Check(K, k, φ, ρ̃ · v) = 1 then
37:       return 1
38:     u ← New(Unrav(K, v, k, FORW))
39:     while u.hasMoreTracks() do
40:       ρ ← u.getNextTrack()
41:       if Check(K, k, φ, ρ̃ · ρ) = 1 then
42:         return 1
43:   return 0
44: else if ψ = ⟨Ē⟩ φ then
45:   for each v ∈ W s.t. (v, fst(ρ̃)) ∈ δ do
46:     if Check(K, k, φ, v · ρ̃) = 1 then
47:       return 1
48:     u ← New(Unrav(K, v, k, BACKW))
49:     while u.hasMoreTracks() do
50:       ρ ← u.getNextTrack()
51:       if Check(K, k, φ, ρ · ρ̃) = 1 then
52:         return 1
53:   return 0
```

The model checking algorithm `ModCheck` requires *exponential working space*, as it uses an instance of the unravelling algorithm and some additional space for a track $\tilde{\rho}$. Analogously, every recursive call to `Check` needs an instance of the unravelling algorithm and space for a track. There are at most $|\psi|$ simultaneously active calls to `Check`, so the total space needed by the considered algorithms is $(|\psi| + 1) \cdot O(|W| + \mathrm{Nest}_{\mathrm{B}}(\psi)) \cdot \tau(|W|, \mathrm{Nest}_{\mathrm{B}}(\psi))$ bits overall, where $\tau(|W|, \mathrm{Nest}_{\mathrm{B}}(\psi))$ is the maximum length of track representatives, and $O(|W| + \mathrm{Nest}_{\mathrm{B}}(\psi))$ bits are needed to represent a state of $\mathcal{K}$, a descriptor element, and a counter for k-indistinguishability.

Notice that formulas ψ of the fragment $HS[A, \overline{A}, \overline{B}, \overline{E}]$ can be checked in polynomial space, as for these formulas $\mathrm{Nest}_{\mathrm{B}}(\psi) = 0$.

We conclude this section by proving that the model checking problem for formulas of $HS[A, \overline{A}, B, \overline{B}, \overline{E}]$, interpreted over finite Kripke structures, is NEXP-hard when a suitable encoding of formulas is exploited. Such an encoding is succinct in the sense that the following binary-encoded shorthands are exploited: $\langle \mathrm{B} \rangle^k \psi$ stands for k repetitions of $\langle \mathrm{B} \rangle$ before ψ, where k is represented in binary (the same for all the other HS modalities); moreover, $\bigwedge_{i=l,\cdots,r} \psi(i)$ denotes a conjunction of formulas which contain some occurrences of the index i as exponents (l and r are binary encoded naturals), e.g., $\bigwedge_{i=1,\cdots,5} \langle \mathrm{B} \rangle^i \top$. Finally, we denote by expand(ψ) the expanded form of ψ, where all exponents k are removed from ψ, by explicitly repeating k times each HS modality with such an exponent, and big conjunctions are replaced by conjunctions of formulas without indexes.

▶ **Theorem 33.** *The model checking problem for $HS[A, \overline{A}, B, \overline{B}, \overline{E}]$ formulas against finite Kripke structures is NEXP-hard, if formulas are succinctly encoded; otherwise, it is NP-hard.*

This result is obtained by means of a reduction from the acceptance problem for a language L decided by a *non-deterministic one-tape* Turing machine M (w.l.o.g.) that halts in $O(2^{n^k})$ computation steps on any input of size n, where $k > 0$ is a constant.

Finally, it is not difficult to show that there exists a constant $c > 0$ such that, for all succinct $HS[A, \overline{A}, B, \overline{B}, \overline{E}]$ formulas ψ, $|\,\mathrm{expand}(\psi)| \leq 2^{|\psi|^c}$. Thus the model checking algorithm still runs in exponential space with respect to the succinct input formula ψ—by preliminarily expanding ψ to $\mathrm{expand}(\psi)$—as $\tau(|W|, \mathrm{Nest_B}(\mathrm{expand}(\psi)))$ is exponential in $|W|$ and $|\psi|$. This allows us to conclude that the model checking problem for succinct $HS[A, \overline{A}, B, \overline{B}, \overline{E}]$ formulas is between NEXP and EXPSPACE.

5 Conclusion and future work

In this paper, we devised an EXPSPACE model checking algorithm for the HS fragments $HS[A, \overline{A}, B, \overline{B}, \overline{E}]$ and $HS[A, \overline{A}, E, \overline{B}, \overline{E}]$ (the known bound for full HS is non-elementary [18]). The algorithm rests on a contraction method that allows us to restrict the verification of the input formula to a finite subset of tracks of bounded size, called track representatives. We also proved that the problem is NEXP-hard, provided that a succinct encoding of formulas is used; otherwise, we can only prove that it is NP-hard (we conjecture that, in this latter case, $HS[A, \overline{A}, B, \overline{B}, \overline{E}]$ is PSPACE-hard). As for the other HS fragments, we showed that $HS[A, \overline{A}, \overline{B}, \overline{E}]$ is in PSPACE, and we conjecture that it is PSPACE-complete. Another interesting fragment is $HS[A, \overline{A}]$ (the logic of temporal neighbourhood): it can be easily shown that it is coNP-hard, but we can only think of PSPACE model checking algorithms.

As for future work, it is worth exploring the model checking problem for full HS and its fragments under other semantic interpretations, relaxing the homogeneity assumption. In this respect, existing work on Duration Calculus (DC) model checking seems to be relevant [7, 8, 10, 12, 15, 21]. DC extends interval temporal logic with an explicit notion of state. States are denoted by state expressions and characterized by a duration (the time period during which the system remains in a given state). Recent results on DC model checking as well as an account of related work can be found in [11].

Acknowledgements. The work by Adriano Peron has been supported by the SHERPA collaborative project, which has received funding from the European Community 7-th Framework Programme (FP7/2007-2013) under grant agreements ICT-600958. He is solely responsible for its content. The paper does not represent the opinion of the European Community and the Community is not responsible for any use that might be made of the information contained therein. The work by Angelo Montanari has been supported by the GNCS project *Algorithms to model check and synthesize safety-critical systems*. We would like to thank the reviewers for their useful comments and suggestions.

References

1 J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

2 D. Bresolin, D. Della Monica, V. Goranko, A. Montanari, and G. Sciavicco. The dark side of interval temporal logic: marking the undecidability border. *Annals of Mathematics and Artificial Intelligence*, 71(1-3):41–83, 2014.

3 D. Bresolin, V. Goranko, A. Montanari, and P. Sala. Tableau-based decision procedures for the logics of subinterval structures over dense orderings. *Journal of Logic and Computation*, 20(1):133–166, 2010.

4 D. Bresolin, V. Goranko, A. Montanari, and G. Sciavicco. Propositional interval neighborhood logics: Expressiveness, decidability, and undecidable extensions. *Annals of Pure and Applied Logic*, 161(3):289–304, 2009.

5 D. Bresolin, A. Montanari, P. Sala, and G. Sciavicco. What's decidable about Halpern and Shoham's interval logic? The maximal fragment $\mathsf{ABB\overline{L}}$. In *LICS*, pages 387–396, 2011.

6 E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 2002.

7 M. Fränzle. Model-checking dense-time Duration Calculus. *Formal Aspects of Computing*, 16(2):121–139, 2004.

8 M. Fränzle and M. R. Hansen. Efficient model checking for Duration Calculus? *International Journal of Software and Informatics*, 3(2-3):171–196, 2009.

9 J. Y. Halpern and Y. Shoham. A propositional modal logic of time intervals. *Journal of the ACM*, 38(4):935–962, 1991.

10 M. R. Hansen. Model-checking discrete Duration Calculus. *Formal Aspects of Computing*, 6(6A):826–845, 1994.

11 M. R. Hansen, A. D. Phan, and A. W. Brekling. A practical approach to model checking Duration Calculus using Presburger Arithmetic. *Annals of Mathematics and Artificial Intelligence*, 71(1-3):251–278, 2014.

12 K. Lodaya. A language-theoretic view of verification. In *Modern Applications of Automata Theory*, pages 149–170, 2012.

13 A. R. Lomuscio and J. Michaliszyn. An epistemic Halpern-Shoham logic. In *IJCAI*, pages 1010–1016, 2013.

14 A. R. Lomuscio and J. Michaliszyn. Decidability of model checking multi-agent systems against a class of EHS specifications. In *ECAI*, pages 543–548, 2014.

15 R. Meyer, J. Faber, J. Hoenicke, and A. Rybalchenko. Model checking Duration Calculus: a practical approach. *Formal Aspects of Computing*, 20(4-5):481–505, 2008.

16 A. Molinari, A. Montanari, A. Murano, G. Perelli, and A. Peron. Checking Interval Properties of Computations. Technical Report 2015/01, Dept. of Math. and CS, University of Udine, 2015. `https://www.dimi.uniud.it/assets/preprints/1-2015-montanari.pdf`.

17 A. Molinari, A. Montanari, and A. Peron. A Model Checking Procedure for Interval Temporal Logics based on Track Representatives. Technical Report 2015/02, Dept. of Math. and CS, University of Udine, 2015. `https://www.dimi.uniud.it/assets/preprints/2-2015-montanari.pdf`.

18 A. Montanari, A. Murano, G. Perelli, and A Peron. Checking interval properties of computations. In *TIME*, pages 59–68, 2014.

19 A. Montanari, G. Puppis, and P. Sala. Maximal decidable fragments of Halpern and Shoham's modal logic of intervals. In *ICALP (2)*, LNCS 6199, pages 345–356, 2010.

20 B. Moszkowski. *Reasoning About Digital Circuits*. PhD thesis, Dept. of Computer Science, Stanford University, Stanford, CA, 1983.

21 P. K. Pandya. Model checking CTL*[DC]. In *TACAS*, LNCS 2031, pages 559–573, 2001.

22 I. Pratt-Hartmann. Temporal prepositions and their logic. *Artificial Intelligence*, 166(1-2):1–36, 2005.

23 P. Roeper. Intervals and tenses. *Journal of Philosophical Logic*, 9:451–469, 1980.

24 Y. Venema. Expressiveness and completeness of an interval tense logic. *Notre Dame Journal of Formal Logic*, 31(4):529–547, 1990.

Contextuality, Cohomology and Paradox

Samson Abramsky[1], Rui Soares Barbosa[1], Kohei Kishida[1], Raymond Lal[1,2], and Shane Mansfield[1]

1 **Department of Computer Science, University of Oxford, Oxford, U.K.**
{samson.abramsky,rui.soares.barbosa,kohei.kishida,shane.mansfield}@cs.ox.ac.uk
2 **Faculty of Philosophy, University of Cambridge, U.K.**
rl335@cam.ac.uk

Abstract

Contextuality is a key feature of quantum mechanics that provides an important non-classical resource for quantum information and computation. Abramsky and Brandenburger used sheaf theory to give a general treatment of contextuality in quantum theory [New Journal of Physics 13 (2011) 113036]. However, contextual phenomena are found in other fields as well, for example database theory. In this paper, we shall develop this unified view of contextuality. We provide two main contributions: first, we expose a remarkable connection between contexuality and logical paradoxes; secondly, we show that an important class of contextuality arguments has a topological origin. More specifically, we show that "All-vs-Nothing" proofs of contextuality are witnessed by cohomological obstructions.

1998 ACM Subject Classification F.1.2 Modes of Computation, F.4.1 Mathematical Logic

Keywords and phrases Quantum mechanics, contextuality, sheaf theory, cohomology, logical paradoxes

Digital Object Identifier 10.4230/LIPIcs.CSL.2015.211

1 Introduction

Contextuality is one of the key characteristic features of quantum mechanics. It has been argued that it provides the "magic" ingredient enabling quantum computation [14]. There have been a number of recent experimental verifications that Nature does indeed exhibit this highly non-classical form of behaviour [33, 32].

The study of quantum contextuality has largely been carried out in a concrete, example-driven fashion, which makes it appear highly specific to quantum mechanics. Recent work by the present authors [3, 5] and others [8] has exposed the general mathematical structure of contextuality, enabling more general and systematic results. It has also made apparent that contextuality is a general and indeed pervasive phenomenon, which can be found in many areas of *classical* computation, such as databases [1] and constraints [4]. The work in [3] makes extensive use of methods developed within the logic and semantics of computation.

The key idea from [3] is to understand contextuality as arising where we have a family of data which is *locally consistent, but globally inconsistent*. This can be understood, and very effectively visualised (see Fig. 3) in topological terms: we have a base space of *contexts* (typically sets of variables which can be jointly measured or observed), a space of data or observations fibred over this space, and a family of *local sections* (typically valuations of the variables in the context) in these fibres. This data is consistent locally, but not globally: there is no *global section* defined on all the variables which reconciles all the local data. In topological language, we can say that the space is "twisted", and hence provides an *obstruction* to forming a global section.

24th EACSL Annual Conference on Computer Science Logic (CSL 2015).
Editor: Stephan Kreutzer; pp. 211–228

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

This provides a unifying description of a number of phenomena which at first sight seem very different:

- *Quantum contextuality.* The local data arises from performing measurements on compatible sets of observables. The fact that there is no global section corresponds to a no-go result for a hidden-variable theory to explain the observable data.
- *Databases.* The local data are the relation tables of the database. The fact that there is no global section corresponds to the failure in general of the *universal relation assumption* [11, 18].
- *Constraint satisfaction.* The local data corresponds to the constraints, defined on subsets of the variables. The fact that there is no global section corresponds to the non-existence of a solution for the CSP.

In the present paper, we shall develop this unified viewpoint to give a logical perspective on contextuality. In particular, we shall look at contextuality in relation to *logical paradoxes*:

- We find a direct connection between the structure of quantum contextuality and classic semantic paradoxes such as "Liar cycles" [10, 30].
- Conversely, contextuality offers a novel perspective on these paradoxes. Contradictory cycles of references give rise to exactly the form of local consistency and global inconsistency we find in contextuality.

Mathematical structure

Sheaf theory [17] provides the natural mathematical setting for our analysis, since it is directly concerned with the passage from local to global. In this setting, it is furthermore natural to use *sheaf cohomology* to characterise contextuality. Cohomology is one of the major tools of modern mathematics, which has until now largely been conspicuous by its absence, both in theoretical computer science, and in quantum information. The use of cohomology to characterise contextuality was initiated in [5]. In the present paper, we take the cohomological approach considerably further, taking advantage of situations in which the outcomes of observations have an algebraic structure. This applies, for example, in the case of the standard Pauli spin observables, which have eigenvalues in $\mathbb{Z}_2$.

We study a strong form of contextuality arising from so-called "All-vs-Nothing" arguments [20]. We give a much more general formulation of such arguments than has appeared previously, in terms of local consistency and global inconsistency of systems of linear equations. We also show how an extensive class of examples of such arguments arises in the stabiliser fragment of quantum mechanics, which plays an important rôle in quantum error correction [23] and measurement-based quantum computation [28].

We then show how all such All-vs-Nothing arguments are witnessed by the cohomological obstruction to the extension of local sections to global ones previously studied in [5]. This obstruction is characterised in more abstract terms than previously, using the connecting homomorphism of the long exact sequence. Our main theorem establishes a hierarchy of properties of probability models, relating their algebraic, logical and topological structures.

For further details and development of the ideas, see the full version of the paper [2].

2 The many faces of contextuality

We begin with the following scenario, depicted in Fig. 1 (a). Alice and Bob are agents positioned at nodes of a network. Alice can access local bit registers a_1 and a_2, while Bob can access local bit registers b_1, b_2. Alice can load one of her bit registers into a processing

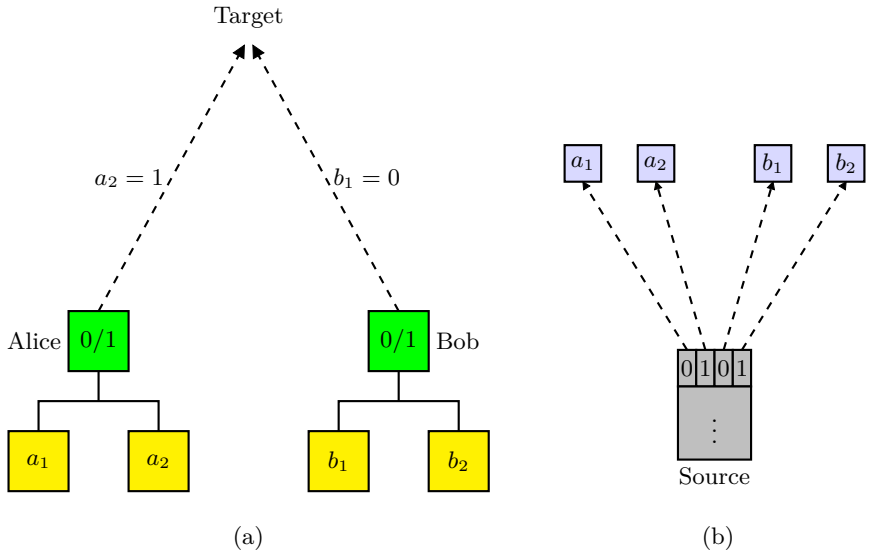


Figure 1 (a) Alice and Bob look at bits. (b) A source.

unit, and test whether it is 0 or 1. Bob can perform the same operations with respect to his bit registers. They send the outcomes of these operations to a common target, which keeps a record of the joint outcomes.

We now suppose that Alice and Bob perform repeated rounds of these operations. On different rounds, they may make different choices of which bit registers to access, and they may observe different outcomes for a given choice of register. The target can compile statistics for this series of data, and infer probability distributions on the outcomes.

2.1 Logical forms of contextuality

While contextuality can exhibit itself at the level of probability distributions (see [3, 2]), here we consider a stronger form of contextuality which exhibits itself at the level of the *supports* of the distributions, highlighting a direct connection with logic.

Consider the tables in Fig. 2, which depict the kind of scenario we have been considering. The entries are either 0 or 1. The idea is that a 1 entry represents a positive probability. Thus we are distinguishing only between *possible* (positive probability) and *impossible* (zero probability). In other words, the rows correspond to the *supports* of some (otherwise unspecified) probability distributions. Note that only four entries of the Hardy table are filled in. Our claim is that just from these entries, referring only to the supports, we can deduce that there is no classical explanation for the behaviour recorded in the table. Moreover, this behaviour can be realised in quantum mechanics [13], yielding a stronger form of Bell's theorem [6], due to Hardy [13].

What do "observables" observe?

Classically, we would take the view that physical observables directly reflect properties of the physical system we are observing. These are objective properties of the system, which are independent of our choice of which measurements to perform, i.e. of our *measurement context*. More precisely, this would say that for each possible state of the system, there is a function λ which for each measurement m specifies an outcome $\lambda(m)$, *independently of*

A	B	$(0,0)$	$(1,0)$	$(0,1)$	$(1,1)$
a_1	b_1	1			
a_1	b_2	0			
a_2	b_1	0			
a_2	b_2				0

A	B	$(0,0)$	$(1,0)$	$(0,1)$	$(1,1)$
a_1	b_1	1	0	0	1
a_1	b_2	1	0	0	1
a_2	b_1	1	0	0	1
a_2	b_2	0	1	1	0

Figure 2 The Hardy paradox (left) and the PR box (right).

which other measurements may be performed. This point of view is called *non-contextuality*, and may seem self-evident. However, this view is *impossible to sustain* in the light of our *actual observations of (micro)-physical reality.*

Consider once again the Hardy table depicted in Fig. 2. Suppose there is a function λ which accounts for the possibility of Alice observing value 0 for a_1 and Bob observing 0 for b_1, as asserted by the entry in the top left position in the table. Then this function λ must satisfy $\lambda(a_1) = 0$, $\lambda(b_1) = 0$. Now consider the value of λ at b_2. If $\lambda(b_2) = 0$, then this would imply that the event that a_1 has value 0 and b_2 has value 0 is possible. However, *this is precluded* by the 0 entry in the table for this event. The only other possibility is that $\lambda(b_2) = 1$. Reasoning similarly with respect to the joint values of a_2 and b_2, we conclude, using the bottom right entry in the table, that we must have $\lambda(a_2) = 0$. Thus the only possibility for λ consistent with these entries is $\lambda :: a_1 \mapsto 0, a_2 \mapsto 0, b_1 \mapsto 0, b_2 \mapsto 1$. But this would require the outcome $(0,0)$ for measurements (a_2, b_1) to be possible, and this is *precluded* by the table.

We are thus forced to conclude that the Hardy models are contextual. Moreover, we can say that they are contextual in a logical sense, stronger than the probabilistic form we saw with the Bell tables, since we only needed information about possibilities to infer the contextuality of this behaviour.

Strong contextuality

Logical contextuality as exhibited by the Hardy paradox can be expressed in the following form: there is a local assignment (in the Hardy case, the assignment $a_1 \mapsto 0, b_1 \mapsto 0$) which is in the support, but which cannot be extended to a global assignment which is compatible with the support. This says that the support cannot be covered by the projections of global assignments. A stronger form of contextuality is when *no global assignments are consistent with the support at all.* Note that this stronger form does not hold for the Hardy paradox.

Several much-studied constructions from the quantum information literature exemplify strong contextuality. An important example is the Popescu–Rohrlich (PR) box [27] shown in Fig. 2.

This is a behaviour which satisfies the *no-signalling principle* [27], meaning that the probability of Alice observing a particular outcome for her choice of measurement (e.g. $a_1 = 0$), is independent of whether Bob chooses measurement b_1 or b_2; and vice versa. That is, Alice and Bob cannot *signal* to one another, enforcing compatibility with relativistic constraints.

In fact, there is provably no bipartite quantum-realisable behaviour of this kind which is strongly contextual [15, 19]. However, as soon as we go to three or more parties, strong contextuality does arise from entangled quantum states, as we shall see in §4.

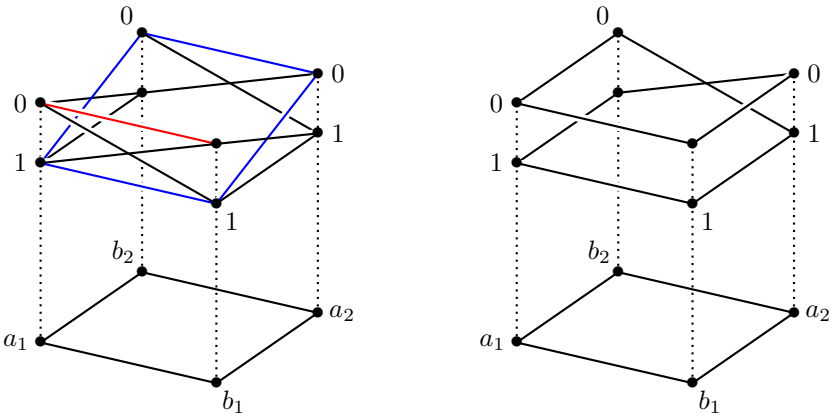


Figure 3 The Hardy table and the PR box as bundles.

Visualizing contextuality

The tables which have appeared in our examples can be displayed in a visually appealing way which makes the fibred topological structure apparent, and forms an intuitive bridge to the formal development of the sheaf-theoretic ideas in the next section.

First, we look at the Hardy table from Fig. 2, displayed as a "bundle diagram" on the left of Fig. 3. Note that all unspecified entries of the Hardy table are set to 1.

What we see in this representation is the *base space* of the variables a_1, a_2, b_1, b_2. There is an edge between two variables when they can be measured together. The pairs of co-measurable variables correspond to the rows of the table. In terms of quantum theory, these correspond to pairs of *compatible observables*. Above each vertex is a *fibre* of those values which can be assigned to the variable – in this example, 0 and 1 in each fibre. There is an edge between values in adjacent fibres precisely when the corresponding *joint outcome* is possible, i.e. has a 1 entry in the table. Thus there are three edges for each of the pairs $\{a_1, b_2\}$, $\{a_2, b_1\}$ and $\{a_2, b_2\}$.

A *global assignment* corresponds to a closed path traversing all the fibres exactly once. We call such a path *univocal* since it assigns a unique value to each variable. Note that there is such a path, marked in blue; thus the Hardy model is not strongly contextual. However, there is no such path which includes the edge displayed in red. This shows the logical contextuality of the model.

Next, we consider the PR box displayed as a bundle on the right of Fig. 3. In this case, the model is strongly contextual, and accordingly there is no univocal closed path.

Contextuality, logic and paradoxes

The arguments for quantum contextuality we have discussed may be said to skirt the borders of paradox, but they do not cross those borders. The information we can gather from observing the co-measurable variables is locally consistent, but it cannot in general be pieced together into a globally consistent assignment of values to all the variables simultaneously. Thus we must give up the idea that physically observable variables have objective, "real" values independent of the measurement context being considered. This is very disturbing for our understanding of the nature of physical reality, but there is no direct contradiction between logic and experience. We shall now show that a similar analysis can be applied to some of the fundamental logical paradoxes.

A *Liar cycle* of length N is a sequence of statements of the following kind.

$$S_1 : S_2 \text{ is true}, \quad S_2 : S_3 \text{ is true}, \quad \ldots \quad , \quad S_{N-1} : S_N \text{ is true}, \quad S_N : S_1 \text{ is false}.$$

For $N = 1$, this is the classic Liar sentence $S : S$ is false. These sentences contain two features which go beyond standard logic: references to other sentences, and a truth predicate. While it would be possible to make a more refined analysis directly modelling these features, we will not pursue this here, noting that it has been argued extensively and rather compellingly in much of the recent literature on the paradoxes that the essential content is preserved by replacing statements with these features by *boolean equations* [31, 10, 30]. For the Liar cycles, we introduce boolean variables $x_1, \ldots, x_n$, and consider the equations $x_1 = x_2$, $\ldots$, $x_{n-1} = x_n$, $x_n = \neg x_1$. The "paradoxical" nature of the original statements is now captured by the inconsistency of these equations.

Note that we can regard each of these equations as fibered over the set of variables which occur in it:

$$\{x_1, x_2\} : x_1 = x_2, \quad \ldots \quad , \quad \{x_{n-1}, x_n\} : x_{n-1} = x_n, \quad \{x_n, x_1\} : x_n = \neg x_1.$$

Any subset of up to $n - 1$ of these equations is consistent; while the whole set is inconsistent.

Up to rearrangement, the Liar cycle of length 4 corresponds exactly to the PR box. The usual reasoning to derive a contradiction from the Liar cycle corresponds precisely to the attempt to find a univocal path in the bundle diagram on the right of Fig. 3. To relate the notations, we make the following correspondences between the variables of Fig. 3 and those of the boolean equations: $x_1 \sim a_2$, $x_2 \sim b_1$, $x_3 \sim a_1$, $x_4 \sim b_2$. Thus we can read the equation $x_1 = x_2$ as "a_2 is correlated with b_1", and $x_4 = \neg x_1$ as "a_2 is anti-correlated with b_2".

Now suppose that we try to set a_2 to 1. Following the path in Fig. 3 on the right leads to the following local propagation of values:

$$a_2 = 1 \rightsquigarrow b_1 = 1 \rightsquigarrow a_1 = 1 \rightsquigarrow b_2 = 1 \rightsquigarrow a_2 = 0$$
$$a_2 = 0 \rightsquigarrow b_1 = 0 \rightsquigarrow a_1 = 0 \rightsquigarrow b_2 = 0 \rightsquigarrow a_2 = 1$$

The first half of the path corresponds to the usual derivation of a contradiction from the assumption that S_1 is true, and the second half to deriving a contradiction from the assumption that S_1 is false.

We have discussed a specific case here, but the analysis can be generalised to a large class of examples along the lines of [10, 30]. The tools from sheaf cohomology which we will develop in the remainder of the paper can be applied to these examples. We plan to give an extended treatment of these ideas in future work.

3 Sheaf formulation of contextuality

In this section we summarise the main ideas of the sheaf-theoretic formalism from [3]. In §2.1 we saw that logical contextuality can be expressed in terms of a bundle of outcomes over a base space of measurements and contexts. This idea can be formalized by regarding the bundle as a sheaf.

For our purposes, it will be sufficient to view the base space as the discrete space on a finite set X of variables.[1] In the quantum case, these variables will be labels for measurements.

[1] The fact that our examples involve a discrete base space X does not trivialise our approach, and

The measurement contexts will be represented by a family $\mathcal{M} = \{C_i\}_{i \in I}$ of subsets of X. These are the sets of variables which can be measured together – in quantum terms, the compatible families of observables. We assume that $\mathcal{M}$ covers X, i.e. $\bigcup \mathcal{M} = X$; hence we call $\mathcal{M}$ a *measurement cover*. We shall also assume that $\mathcal{M}$ forms an antichain, so these are the *maximal contexts*. We also assume that all the variables have the same fibre, O, of values or outcomes that can be assigned to them. Such a triple $\langle X, \mathcal{M}, O \rangle$ is called a *measurement scenario*. We define a presheaf of sets over $\mathcal{P}(X)$, namely $\mathcal{E} :: U \longmapsto O^U$ with restriction $\mathcal{E}(U \subseteq U') \colon \mathcal{E}(U') \longrightarrow \mathcal{E}(U) :: s \longmapsto s|_U$. This presheaf $\mathcal{E}$ is in fact a sheaf, called the *sheaf of events*. Each $s \in \mathcal{E}(U)$ is a *section*, and, in particular, $g \in \mathcal{E}(X)$ is a *global section*.

Note that a probability table can be represented by a family $\{p_C\}_{C \in \mathcal{M}}$ with p_C a probability distribution on $\mathcal{E}(C) = O^C$, where contexts C correspond to the rows of the table. Similarly, "possibility tables" such as the Hardy model and the PR box (Figs. 2 and 3) can be represented by boolean distributions. This latter case, with which the logical and strong forms of contextuality are concerned, can equivalently be represented by a subpresheaf $\mathcal{S}$ of $\mathcal{E}$, where for each context $U \subseteq X$, $\mathcal{S}(U) \subseteq O^U$ is the set of all possible outcomes. Explicitly, $\mathcal{S}$ is defined as follows, where $\mathsf{supp}\,(p_C|_{U \cap C})$ is the support of the marginal of p_C at $U \cap C$.

$$\mathcal{S}(U) := \left\{ s \in O^U \;\middle|\; \forall C \in \mathcal{M}.\; s|_{U \cap C} \in \mathsf{supp}\,(p_C|_{U \cap C}) \right\}$$

Abstracting from this situation, we assume we are dealing with a sub-presheaf $\mathcal{S}$ of $\mathcal{E}$ with the following properties:

E1. $\mathcal{S}(C) \neq \varnothing$ for all $C \in \mathcal{M}$
(i.e. that any possible joint measurement yields some joint outcome), and moreover that

E2. $\mathcal{S}$ is *flasque beneath the cover*, meaning that $\mathcal{S}(U \subseteq U') \colon \mathcal{S}(U') \longrightarrow \mathcal{S}(U)$ is surjective whenever $U \subseteq U' \subseteq C$ for some $C \in \mathcal{M}$,
which by [3] amounts to saying that the underlying empirical model satisfies no-signalling.

E3. A *compatible family* for the cover $\mathcal{M}$ is a family $\{s_C\}_{C \in \mathcal{M}}$ with $s_C \in \mathcal{S}(C)$, and such that, for all $C, C' \in \mathcal{M}$: $s_C|_{C \cap C'} = s_{C'}|_{C \cap C'}$. We assume that such a family induces a global section in $\mathcal{S}(X)$. (This global section must be unique, since $\mathcal{S}$ is a subpresheaf of $\mathcal{E}$, hence separated).

What these conditions say is that $\mathcal{S}$ is determined by its values $\mathcal{S}(C)$ at the contexts $C \in \mathcal{M}$, below $\mathcal{M}$ by being flasque, and above $\mathcal{M}$ by the sheaf condition.

▶ **Definition 1.** By an *empirical model* on $\langle X, \mathcal{M}, O \rangle$, we mean a subpresheaf $\mathcal{S}$ of $\mathcal{E}$ satisfying (E1), (E2), and (E3).

In [3], we used the term "empirical model" for the probability table $\{p_C\}_{C \in \mathcal{M}}$. In the present paper, we shall only work with the associated support presheaf $\mathcal{S}$, and so it is more convenient to refer to this as the model.

We can use this formalisation to characterize contextuality as follows.

▶ **Definition 2.** For any empirical model $\mathcal{S}$:

- For $C \in \mathcal{M}$ and $s \in \mathcal{S}(C)$, $\mathcal{S}$ is logically contextual at s, written $\mathsf{LC}(\mathcal{S}, s)$, if s belongs to no compatible family. $\mathcal{S}$ is *logically contextual*, written $\mathsf{LC}(\mathcal{S})$, if $\mathsf{LC}(\mathcal{S}, s)$ for some s.

certainly does not mean that we are taking the cohomology of a discrete space! It is standard that in a topological bundle, the interesting twisting occurs in the fibres, not in the base. A classic example is the Möbius strip, displayed as a fibre bundle over the circle. The circle is not twisted! In our case, it is clear from our examples that non-trivial twisting does occur. Moreover, our results in Section 6 will clearly show the non-triviality of our cohomological obstructions.

- $\mathcal{S}$ is *strongly contextual*, written $\mathsf{SC}(\mathcal{S})$, if $\mathsf{LC}(\mathcal{S}, s)$ for all s. Equivalently, it is strongly contextual if it has no global section, i.e. if $\mathcal{S}(X) = \varnothing$.

Note that, for every probability table $\{p_C\}_{C \in \mathcal{M}}$ that satisfies the no-signalling principle, the supports of the distributions p_C induce an empirical model $\mathcal{S}$, and therefore logical or strong contextuality can be characterized as above. This formulation of contextuality makes it natural to use sheaf cohomology, as we will see in §5.

4 All-vs-Nothing arguments

Quantum theory provides many instances of strong contextuality. Among the first to observe quantum strong contextuality (though not in the general terms that we do) was Mermin [21], who showed the GHZ state to be strongly contextual using a kind of argument he dubbed 'all versus nothing'. We show in §4.1 that arguments of this type can in fact be used to prove strong contextuality for a large class of states in quantum theory, particularly in stabiliser quantum mechanics, which plays a crucial rôle in quantum computation. Moreover, in §4.2, we give a much more general formulation of this type of argument that can be used to show strong contextuality for a much larger class of models.

4.1 All-vs-Nothing for quantum theory

The GHZ state is a tripartite state of qubits, defined as $(|\uparrow\uparrow\uparrow\rangle + |\downarrow\downarrow\downarrow\rangle)/\sqrt{2}$. We assume that each party $i = 1, 2, 3$ can perform Pauli measurements in $\{X_i, Y_i\}$, and each measurement has outcomes in $O = \mathbb{Z}_2 = \{0, 1\}$.[2] Then, following Mermin's argument, the possible joint outcomes satisfy these parity equations:

$$X_1 \oplus Y_2 \oplus Y_3 = 1, \quad Y_1 \oplus Y_2 \oplus X_3 = 1, \quad Y_1 \oplus X_2 \oplus Y_3 = 1, \quad X_1 \oplus X_2 \oplus X_3 = 0.$$

These equations are inconsistent because, regardless of the outcomes assigned to the observables $X_1, \ldots, Y_3$, the left-hand sides sum to 0 (since each variable occurs twice) whereas the right-hand sides sum to 1. This shows that the model is strongly contextual, as there is no global assignment of outcomes to observables consistent with the observed local assignments.

The essence of the argument is that the possible local assignments satisfy systems of parity equations that admit no global solution. We call this an *All-vs-Nothing argument*.

In fact, such arguments arise naturally from a much larger class of states in stabiliser quantum theory [23]. Consider the Pauli n-group $\mathcal{P}_n$, whose elements are n-tuples of Pauli operators (from $\{X, Y, Z, I\}$) with a global phase from $\{\pm 1, \pm i\}$.

▶ **Definition 3.** An *AvN triple* in $\mathcal{P}_n$ is a triple $\langle e, f, g \rangle$ of elements of $\mathcal{P}_n$ with global phases $+1$, which pairwise commute, and which satisfy the following conditions:
A1. For each $i = 1, \ldots, n$, at least two of e_i, f_i, g_i are equal.
A2. The number of i such that $e_i = g_i \neq f_i$, all distinct from I, is odd.

Mermin's argument, and the other All-vs-Nothing arguments which have appeared in the literature, can be seen to come down to exhibiting AvN triples.

[2] Although the eigenvalues of the Pauli matrices are $+1$ and -1, we relabel $+1$, -1, $\times$ as 0, 1, $\oplus$, respectively. The eigenvalues of a joint measurements $A_1 \otimes A_2 \otimes A_3$ are the products of the eigenvalues of the measurements at each site, so they are also ± 1. As such, in the usual representation, these joint measurements are still dichotomic and only distinguish joint outcomes up to parity. Mermin's argument shows that this information is sufficient to derive strong contextuality.

▶ **Theorem 4.** *Let S be the subgroup of $\mathcal{P}_n$ generated by an AvN triple, and V_S the subspace stabilised by S. For every state $|\psi\rangle$ in V_S, the empirical model realised by $|\psi\rangle$ under the Pauli measurements admits an All-vs-Nothing argument.*

Proof. First, we recall the quantum mechanics formula for the expected value of an observable A on a state v:

$$\langle A\rangle_v \;=\; \langle v|A|v\rangle.$$

Note that

$$\langle v|A|v\rangle = 1 \iff A|v\rangle = |v\rangle.$$

Thus A stabilises the state v iff the expected value is 1. Suppose that A is a dichotomic observable, with eigenvalues $+1$, -1 (see footnote 3), and v is a state it stabilises. The support of the distribution on joint outcomes obtained by measuring A on v must contain only outcomes of even parity; while if $-A$ stabilises v, then the support will contain only outcomes of odd parity. If $A = P_1 \cdots P_n$ in $\mathcal{P}_n$, the former case translates into the equation

$$x_1 \oplus \cdots \oplus x_n = 0$$

where we associate the variable x_i with P_i; while in the latter case, it corresponds to the equation

$$x_1 \oplus \cdots \oplus x_n = 1.$$

If the set of equations satisfied by a state v stabilised by a subgroup S of $\mathcal{P}_n$ is inconsistent, we say that v *admits an All-vs-Nothing argument* with respect to the measurements h with global phase $+1$ such that either h or $-h$ is in S.

We now show that any state v in the subspace V_S stabilised by the subgroup S generated by an AvN triple $\langle e, f, g\rangle$ admits an All-vs-Nothing argument. First, by the algebra of the Pauli matrices, we see from (A1) that if $\{e_i, f_i, g_i\} = \{P, Q\}$, with at least two equal to P, the componentwise product $e_i f_i g_i$ will, disregarding global phase, be Q. By (A2), we see that the product $efg = -h$, an element of $\mathcal{P}_n$ with global phase -1, which translates into a condition of odd parity on the support of any state stabilised by these operators for the measurement h. On the other hand, condition (A1) implies that under any global assignment to the variables, we can cancel the repeated items in each column, and deduce an even parity for h. ◀

If e, f, g have linearly independent check vectors, they generate a subgroup S such that V_S has dimension 2^{n-3} [23, 9]. Thus we obtain a large class of states admitting All-vs-Nothing arguments.

4.2 Generalized All-vs-Nothing arguments

Despite their established use in the quantum literature, All-vs-Nothing arguments as considered above do not exist for all strongly contextual models. However, a natural generalization applies to more models. For instance, the model called 'box 25' in [26] admits no parity AvN argument, but it still satisfies the following equations, in which the coefficients of each variable on the left-hand sides add up to a multiple of 3, whereas the right-hand sides do not:

$$\begin{aligned} a_0 + 2b_0 &\equiv 0 \bmod 3 & a_1 + 2c_0 &\equiv 0 \bmod 3 \\ a_0 + b_1 + c_0 &\equiv 2 \bmod 3 & a_0 + b_1 + c_1 &\equiv 2 \bmod 3 \\ a_1 + b_0 + c_1 &\equiv 2 \bmod 3 & a_1 + b_1 + c_1 &\equiv 2 \bmod 3 \end{aligned}$$

This example suggests using a general $\mathbb{Z}_n$ instead of just $\mathbb{Z}_2$. But once we realize that it is the ring structure of $\mathbb{Z}_n$ which plays the key rôle, we can obtain an even more general version.

Fix a ring[3] R, and a measurement scenario $\langle X, \mathcal{M}, R \rangle$.

▶ **Definition 5.** An *R-linear equation* is a triple $\phi = \langle C, a, b \rangle$ with $C \in \mathcal{M}$, $a \colon C \longrightarrow R$ and $b \in R$. Write $V_\phi := C$. An assignment $s \in \mathcal{E}(C)$ *satisfies* ϕ, written $s \models \phi$, if

$$\sum_{m \in C} a(m)s(m) = b \ .$$

This lifts to the level of systems of equations, or theories, and sets of assignments, or "models":

- A system of equations Γ has a set of satisfying assignments, $\mathbb{M}(\Gamma) := \{s \in \mathcal{E}(C) \mid \forall \phi \in \Gamma \mathbf{.}\ s \models \phi\}$.
- A set of assignments $S \subseteq \mathcal{E}(C)$ determines an R-linear theory, $\mathbb{T}_R(S) := \{\phi \mid \forall s \in S \mathbf{.}\ s \models \phi\}$.

▶ **Definition 6.** Given an empirical model $\mathcal{S}$, define its *R-linear theory* to be

$$\mathbb{T}_R(\mathcal{S}) := \bigcup_{C \in \mathcal{M}} \mathbb{T}_R(\mathcal{S}(C)) = \{\phi \mid \forall s \in \mathcal{S}(V_\phi) \mathbf{.}\ s \models \phi\} \ .$$

We say that $\mathcal{S}$ is AvN_R, written $\mathsf{AvN}_R(\mathcal{S})$, if $\mathbb{T}_R(\mathcal{S})$ is inconsistent, meaning that there is no global assignment $g \colon X \longrightarrow R$ such that $\forall \phi \in \mathbb{T}_R(\mathcal{S}) \mathbf{.}\ g|_{V_\phi} \models \phi$.

▶ **Proposition 7.** *An AvN_R model is strongly contextual.*

Proof. Suppose $\mathcal{S}$ is not strongly contextual, i.e. that there is some $g \in \mathcal{S}(X)$. Then, for each $\phi \in \mathbb{T}_R(\mathcal{S})$, $g|_{V_\phi} \in \mathcal{S}(V_\phi)$, hence $g|_{V_\phi} \models \phi$. Thus, $\mathbb{T}_R(\mathcal{S})$ is consistent. ◀

4.3 Affine closures

We now consider the relationship between R-linear theories and empirical models more closely. First, we focus on a single context, or set of variables, $U \subseteq X$. The maps between theories and models, $\mathbb{T} : \mathcal{PE}(U) \rightleftarrows \mathsf{Theories} : \mathbb{M}$, form a Galois connection, $S \subseteq \mathbb{M}(\Gamma)$ iff $\mathbb{T}(S) \supseteq \Gamma$, corresponding to the lifting of the satisfaction relation to the powersets.

We consider the closure operator $\mathbb{M} \circ \mathbb{T}$, which gives the largest set of assignments whose theory is still the same. First, note that $\mathcal{E}(U) = R^U$ is a (free) R-module (and, when R is a field, a vector space over R). Given solutions $s_1, \ldots, s_t$ to a linear equation, an affine combination of them is again a solution[4]. In other words, the set of solutions $\mathbb{M}(\Gamma)$ to a system of equations Γ is an affine submodule of $\mathcal{E}(U)$. This means that $\mathsf{aff} \leq \mathbb{M} \circ \mathbb{T}$, where $\mathsf{aff}\, S$ stands for the *affine closure* of a set $S \subseteq \mathcal{E}(U)$:

$$\mathsf{aff}\ S \ := \left\{ \sum_{i=1}^{t} c_i s_i \;\middle|\; s_i \in S, c_i \in R, \sum_{i=1}^{t} c_i = 1 \right\} \ .$$

In the particular case of vector spaces (i.e. when R is a field), then $\mathsf{aff} = \mathbb{M} \circ \mathbb{T}$; affine subspaces are exactly the possible solution sets of a theory, and there cannot exist two different affine subspaces with the same theory, as may happen for general rings R.

We now lift this discussion to the level of empirical models. The natural thing to do is to take the affine closure at each context $C \in \mathcal{M}$. However, one must be careful to ensure that

[3] All rings considered in this paper will be commutative and with unit.
[4] Affineness is required because the equations may be inhomogeneous.

this yields a well-defined empirical model. First, note that the affine closure operation above is natural on U: it gives a natural transformation $\mathsf{aff} : \mathcal{P} \circ \mathcal{E} \longrightarrow \mathcal{P} \circ \mathcal{E}$, meaning that

$$(\mathsf{aff}\, S)|_{U'} = \mathsf{aff}\,(S|_{U'})\ . \tag{1}$$

This follows easily from the coordinatewise definitions of the module operations on $\mathcal{E}(U)$.

▶ **Definition 8.** Let $\mathcal{S}$ be an empirical model on the scenario $\langle X, \mathcal{M}, R\rangle$. We define its *affine closure*, $\mathsf{Aff}\,\mathcal{S}$, as the empirical model given by $(\mathsf{Aff}\,\mathcal{S})(C) := \mathsf{aff}\,(\mathcal{S}(C))$ at each $C \in \mathcal{M}$.

The property (1) guarantees that $\mathsf{Aff}\,\mathcal{S}$ can be consistently defined to be flasque below the cover as $(\mathsf{Aff}\,\mathcal{S})(U) = \mathsf{aff}\,(\mathcal{S}(U))$. This equality, however, does not hold for U above the cover. In particular, it may be that $\mathcal{S}(X) = \varnothing$ ($\mathcal{S}$ strongly contextual), but $(\mathsf{Aff}\,\mathcal{S})(X) \neq \varnothing$.

Since $\mathbb{T}_R(\mathcal{S})$ is given as the union of the theories at each maximal context, the Galois connection above lifts to the level of empirical models. One can therefore relate the notion of $\mathcal{S}$ being AvN_R to the strong contextuality of the affine closure of $\mathcal{S}$.

▶ **Proposition 9.** *Let $\mathcal{S}$ be an empirical model on $\langle X, \mathcal{M}, R\rangle$. Then, $\mathsf{AvN}(\mathcal{S}) \Rightarrow \mathsf{SC}(\mathsf{Aff}\,\mathcal{S})$. If R is a field, the converse also holds.*

Proof. From $\mathsf{aff} \leq \mathbb{M} \circ \mathbb{T}$, $\mathbb{T}_R(\mathcal{S}) = \mathbb{T}_R(\mathsf{Aff}\,\mathcal{S})$. Hence, if $\mathcal{S}$ is AvN_R, then so is $\mathsf{Aff}\,\mathcal{S}$, implying by Proposition 7 that it is strongly contextual.

For the converse in the case that R is a field, suppose that $\mathbb{T}_R(\mathcal{S})$ is consistent. This means that there is a global assignment $g\colon X \longrightarrow R$ satisfying all the equations in $\mathbb{T}_R(\mathcal{S})$. But since for fields $\mathbb{MT}_R(\mathcal{S}) = \mathsf{Aff}\,\mathcal{S}$, we have that $g \in (\mathsf{Aff}\,\mathcal{S})(X)$, hence the model $\mathsf{Aff}\,\mathcal{S}$ is not strongly contextual. ◀

5 Cohomology witnesses contextuality

The logical forms of contextuality are characterised by the existence of obstructions to the extension of local sections to global compatible families. Thus, it seems natural to apply the tools of sheaf cohomology, which are well-suited to identifying obstructions of this kind, in order to provide cohomological witnesses for contextuality. This idea was put forward in previous work by the authors [5], the main points of which we now summarise. In the next section, we shall prove that such cohomological witnesses of contextuality exist for the whole class of AvN_R models.

5.1 Čech cohomology

Let X be a topological space, $\mathcal{M}$ be an open cover of X, and let $\mathcal{F}\colon \mathcal{O}(X)^{\mathsf{op}} \longrightarrow \mathsf{AbGrp}$ be a presheaf of abelian groups on X. We shall be particularly concerned with the case where X is a set of measurements, and $\mathcal{M}$ is the cover of maximal contexts of a measurement scenario.

▶ **Definition 10.** A *q-simplex of the nerve* of $\mathcal{M}$, is a tuple $\sigma = \langle C_0, \ldots, C_q\rangle$ of elements of $\mathcal{M}$ with non-empty intersection, $|\sigma| := \cap_{i=0}^{q} C_i \neq \varnothing$. We write $\mathcal{N}(\mathcal{M})^q$ for the set of q-simplices.

The 0-simplices are simply the elements of the cover $\mathcal{M}$, and the 1-simplices are the pairs $\langle C_i, C_j\rangle$ of intersecting elements of the cover. Given a $q+1$-simplex $\sigma = \langle C_0, \ldots, C_{q+1}\rangle$, we can obtain q-simplices

$$\partial_j(\sigma) := \langle C_0, \ldots, \widehat{C_j}, \ldots, C_{q+1}\rangle, \qquad 0 \leq j \leq q+1$$

by omitting one of the elements. Note that $|\sigma| \subseteq |\partial_j(\sigma)|$, and so the presheaf $\mathcal{F}$ has a restriction map $\rho_{|\sigma|}^{|\partial_j(\sigma)|}\colon \mathcal{F}(|\partial_j(\sigma)|) \longrightarrow \mathcal{F}(|\sigma|)$. We now define the *Čech cochain complex*.

▶ **Definition 11.** For each $q \geq 0$, the abelian group of *q-cochains* is defined by:

$$C^q(\mathcal{M}, \mathcal{F}) \;:=\; \prod_{\sigma \in \mathcal{N}(\mathcal{M})^q} \mathcal{F}(|\sigma|) \;.$$

The *q-coboundary map*, $\delta^q \colon C^q(\mathcal{M}, \mathcal{F}) \longrightarrow C^{q+1}(\mathcal{M}, \mathcal{F})$, is defined as follows: for each $\omega = (\omega(\tau))_{\tau \in \mathcal{N}(\mathcal{M})^q} \in C^q(\mathcal{M}, \mathcal{F})$, and $\sigma \in \mathcal{N}(\mathcal{M})^{q+1}$,

$$\delta^q(\omega)(\sigma) \;:=\; \sum_{j=0}^{q+1} (-1)^j \rho^{|\partial_j(\sigma)|}_{|\sigma|} \omega(\partial_j \sigma) \;.$$

The *augmented Čech cochain complex* is the sequence

$$\mathbf{0} \longrightarrow C^0(\mathcal{M}, \mathcal{F}) \longrightarrow C^1(\mathcal{M}, \mathcal{F}) \longrightarrow \cdots \;.$$

▶ **Proposition 12.** *For each q, δ^q is a group homomorphism, and we have $\delta^{q+1} \circ \delta^q = 0$.*

▶ **Definition 13.** For each $q \geq 0$, we define:

- the *q-cocycles* $Z^q(\mathcal{M}, \mathcal{F}) := \mathsf{ker}\, \delta^q$;
- the *q-coboundaries* $B^q(\mathcal{M}, \mathcal{F}) := \mathsf{im}\, \delta^{q-1}$.

By Proposition 12, these are subgroups of $C^q(\mathcal{M}, \mathcal{F})$ with $B^q(\mathcal{M}, \mathcal{F}) \subseteq Z^q(\mathcal{M}, \mathcal{F})$.

▶ **Definition 14.** $\check{H}^q(\mathcal{M}, \mathcal{F})$, the *q-th Čech cohomology group*, is defined as the quotient $Z^q(\mathcal{M}, \mathcal{F})/B^q(\mathcal{M}, \mathcal{F})$.

Note that $B^0(\mathcal{M}, \mathcal{F}) = \mathbf{0}$, and hence $\check{H}^0(\mathcal{M}, \mathcal{F}) \cong Z^0(\mathcal{M}, \mathcal{F})$. A 0-cochain ω is a family $\{r_C \in \mathcal{F}(C)\}_{C \in \mathcal{M}}$. Since, for each 1-simplex $\sigma = (C, C')$,

$$\delta^0(\omega)(\sigma) \;=\; r_C|_{C \cap C'} \;-\; r_{C'}|_{C \cap C'} \;,$$

ω is a cocycle (i.e. satisfies $\delta^0(c) = 0$) if and only if $r_C|_{C \cap C'} = r_{C'}|_{C \cap C'}$ for all maximal contexts $C, C' \in \mathcal{M}$ with non-empty intersection.[5]

5.2 Relative cohomology and obstructions

In order to solve the problem of extending a local section to a global compatible family, we need to consider the relative cohomology of $\mathcal{F}$ with respect to an open subset $U \subseteq X$. We will assume that the presheaf is flasque beneath the cover (as is the case with $\mathcal{S}$).

We define two auxiliary presheaves related to $\mathcal{F}$. First, $\mathcal{F}|_U$ is defined by

$$\mathcal{F}|_U(V) := \mathcal{F}(U \cap V) \;.$$

---

[5] The condition for a 0-cochain $\omega = \{r_C\}$ to be a cocycle almost states that r is a compatible family, except that it does not require compatibility over restrictions to the empty context. For our present purposes, we are only interested in connected covers (since one can always reduce the analysis of a scenario to its connected components), in which case the exception is irrelevant. This is because, given any two contexts C and C' with empty intersection, there exists a sequence of contexts

$$C = C_0,\; C_1,\; \ldots,\; C_n = C'$$

such that $C_i \cap C_{i+1} \neq \varnothing$ for all i. Then, for any i, we have

$$r_{C_i}|_\varnothing = r_{C_i}|_{C_i \cap C_{i+1}}|_\varnothing = r_{C_{i+1}}|_{C_i \cap C_{i+1}}|_\varnothing = r_{C_{i+1}}|_\varnothing \;.$$

Consequently, $r_C|_{C \cap C'} = r_C|_\varnothing = r_{C'}|_\varnothing = r_{C'}|_{C \cap C'}$, and so the family is compatible.

There is an evident presheaf map $p\colon \mathcal{F} \longrightarrow \mathcal{F}|_U$ given as

$$p_V : \mathcal{F}(V) \longrightarrow \mathcal{F}(U \cap V) :: r \longmapsto r|_{U\cap V} \ .$$

Secondly, $\mathcal{F}_{\bar{U}}$ is defined by $\mathcal{F}_{\bar{U}}(V) := \mathsf{ker}\,(p_V)$. Thus, we have an exact sequence of presheaves

$$\mathbf{0} \longrightarrow \mathcal{F}_{\bar{U}} \longrightarrow \mathcal{F} \xrightarrow{p} \mathcal{F}|_U \quad . \tag{2}$$

The *relative cohomology of $\mathcal{F}$ with respect to U* is defined to be the cohomology of the presheaf $\mathcal{F}_{\bar{U}}$.

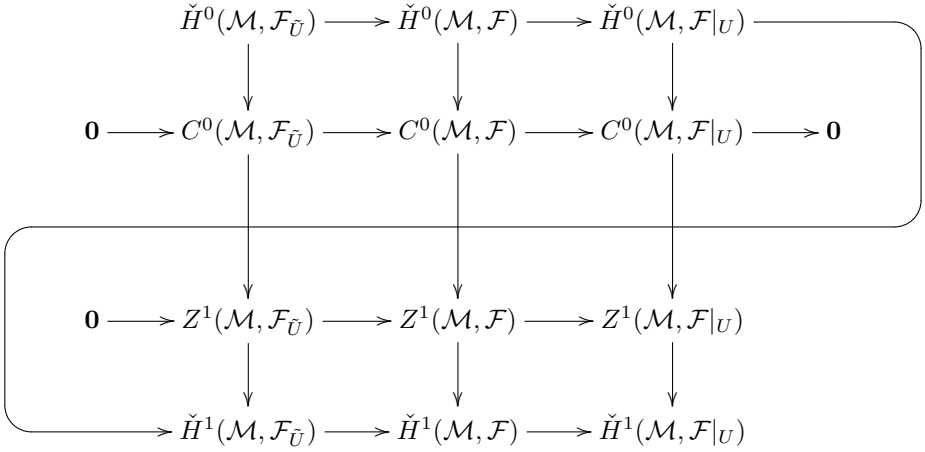
We now see how this can be used to identify *cohomological obstructions* to the extension of a local section. First, recall that in light of Proposition 12, the image of δ^0, $B^1(\mathcal{M},\mathcal{F})$, is contained in $Z^1(\mathcal{M},\mathcal{F})$. Therefore, the map δ^0 can be corestricted to a map $\tilde{\delta}^0\colon C^0(\mathcal{M},\mathcal{F}) \longrightarrow Z^1(\mathcal{M},\mathcal{F})$, whose kernel is $Z^0(\mathcal{M},\mathcal{F}) \cong \check{H}^0(\mathcal{M},\mathcal{F})$ and whose cokernel is $Z^1(\mathcal{M},\mathcal{F})/B^1(\mathcal{M},\mathcal{F}) \cong \check{H}^1(\mathcal{M},\mathcal{F})$. In summary, we have:

$$\check{H}^0(\mathcal{M},\mathcal{F}) \xrightarrow{\mathsf{ker}\,\tilde{\delta}^0} C^0(\mathcal{M},\mathcal{F}) \xrightarrow{\tilde{\delta}^0} Z^1(\mathcal{M},\mathcal{F}) \xrightarrow{\mathsf{coker}\,\tilde{\delta}^0} \check{H}^1(\mathcal{M},\mathcal{F}) \quad .$$

We now lift the exact sequence of presheaves (2) considered above to the level of cochains. The map $C^0(\mathcal{M},\mathcal{F}) \longrightarrow C^0(\mathcal{M},\mathcal{F}|_U)$ is surjective due to flaccidity beneath the cover. Putting this together with the previous observation, we obtain the diagram below:

$$\begin{array}{ccccccccc}
\mathbf{0} & \longrightarrow & C^0(\mathcal{M},\mathcal{F}_{\tilde{U}}) & \longrightarrow & C^0(\mathcal{M},\mathcal{F}) & \longrightarrow & C^0(\mathcal{M},\mathcal{F}|_U) & \longrightarrow & \mathbf{0} \\
 & & \downarrow{\scriptstyle \tilde{\delta}^0} & & \downarrow{\scriptstyle \tilde{\delta}^0} & & \downarrow{\scriptstyle \tilde{\delta}^0} & & \\
\mathbf{0} & \longrightarrow & Z^1(\mathcal{M},\mathcal{F}_{\tilde{U}}) & \longrightarrow & Z^1(\mathcal{M},\mathcal{F}) & \longrightarrow & Z^1(\mathcal{M},\mathcal{F}|_U) & &
\end{array}$$

whose two rows are short exact sequences. The *snake lemma* of homological algebra says that there exists a *connecting homomorphism* turning the kernels of the first row followed by the cokernels of the second into a long exact sequence, as shown by the following diagram.



We are interested in the case where U is an element C_0 of the cover $\mathcal{M}$. Then $\check{H}^0(\mathcal{M},\mathcal{F}|_{C_0})$ is clearly isomorphic to $\mathcal{F}(C_0)$, meaning that its elements are the local sections at C_0.

▶ **Definition 15.** Let C_0 be an element of the cover $\mathcal{M}$ and $r_0 \in \mathcal{F}(C_0)$. Then, the *cohomological obstruction* of r_0 is the element $\gamma(r_0)$ of $\check{H}^1(\mathcal{M},\mathcal{F}_{\tilde{C_0}})$, where $\gamma\colon \check{H}^0(\mathcal{M},\mathcal{F}|_U) \longrightarrow \check{H}^1(\mathcal{M},\mathcal{F}_{\tilde{U}})$ is the connecting homomorphism.

The following proposition justifies regarding these as obstructions.

▶ **Proposition 16.** *Let the cover $\mathcal{M}$ be connected, $C_0 \in \mathcal{M}$, and $r_0 \in \mathcal{F}(C_0)$. Then, $\gamma(r_0) = 0$ if and only if there is a compatible family $\{r_C \in \mathcal{F}(C)\}_{C \in \mathcal{M}}$ such that $r_{C_0} = r_0$.*

For a proof of this proposition, see [2].

▶ Remark. Note that our cohomology obstruction lives in the first cohomology group. In ordinary homology, the lower groups capture low-dimensional behaviour, and to capture higher-dimensional behaviour, one must pass to the higher homology groups. The situation is quite different in sheaf cohomology; there, it is standard that it is the first cohomology group which captures obstructions to extending local sections to global ones. Of course, it would also be of interest to find natural uses for the higher cohomology groups.

5.3 Witnessing contextuality

We now apply these tools to analyse the possibilistic structure of empirical models. The cohomological obstructions of Definition 15 would appear to be ideally suited to the problem of identifying contextuality. The caveat is that, in order to apply those tools, it is necessary to work over a presheaf of abelian groups, whereas we are concerned with $\mathcal{S}$, which is merely a presheaf of sets. We first consider how to build an abelian group from a set.

▶ **Definition 17.** Given a ring R, we define a functor $F_R \colon \mathsf{Set} \longrightarrow R\text{-}\mathsf{Mod}$ to the category of R-modules (and thus, in particular, to the category of abelian groups). For each set X, $F_R(X)$ is the set of functions $\phi \colon X \longrightarrow R$ of finite support. Given a function $f \colon X \longrightarrow Y$,

$$F_R f \colon F_R X \longrightarrow F_R Y :: \phi \longmapsto \lambda y. \sum_{f(x)=y} \phi(x) \ .$$

This is easily seen to be functorial. We regard a function $\phi \in F_R(X)$ as a *formal R-linear combination* of elements of X: $\sum_{x \in X} \phi(x) \cdot x$. There is a natural embedding $x \mapsto 1 \cdot x$ of X into $F_R(X)$, which we shall use implicitly throughout. In fact, $F_R(X)$ is the *free R-module generated by X*; and in particular, $F_{\mathbb{Z}}(X)$ is the *free abelian group generated by X*.

Given an empirical model $\mathcal{S}$ defined on the measurement scenario $\langle X, \mathcal{M}, O \rangle$, we shall work with the (relative) Čech cohomology for the abelian presheaf $F_R\mathcal{S}$ for some ring R.

▶ **Definition 18.** With each local section, $s \in \mathcal{S}(C)$, in the support of an empirical model, we associate the *cohomological obstruction* $\gamma_{F_R\mathcal{S}}(s)$.

- If there exists some local section $s_0 \in \mathcal{S}(C_0)$ such that $\gamma_{F_R\mathcal{S}}(s_0) \neq 0$, we say that $\mathcal{S}$ is *cohomologically logically contextual*, or $\mathsf{CLC}_R(\mathcal{S})$. We also use the more specific notation $\mathsf{CLC}_R(\mathcal{S}, s_0)$.
- If $\gamma_{F_R\mathcal{S}}(s) \neq 0$ for all local sections, we say that e is *cohomologically strongly contextual*, or CSC_R.

The following justifies considering cohomological obstructions as witnessing contextuality.

▶ **Proposition 19** ([5, Proposition 4.3])**.** CLC_R *implies* LC*, and* CSC_R *implies* SC.

Proof. Suppose an empirical model e is not logically contextual. Then for every maximal context $C_0 \in \mathcal{M}$ and every $s_0 \in \mathcal{S}(C_0)$, there is a compatible family $\{s_C \in \mathcal{S}(C)\}_{C \in \mathcal{M}}$ with $s_{c_0} = s_0$. As $\mathcal{S}(C)$ embeds into $F_R\mathcal{S}(C)$, $\{s_C\}$ is also a compatible family in $F_R\mathcal{S}$. Hence, by Proposition 16, we conclude that $\gamma(s) = 0$. The same argument can be applied to a single section witnessing the failure of strong contextuality. ◀

Thus we have a sufficient condition for contextuality in the existence of a cohomological obstruction. Unfortunately, this condition is not, in general, necessary. It is possible that "false positives" arise in the form of families $\{r_C \in F_R\mathcal{S}(C)\}_{C\in\mathcal{M}}$ which are not *bona fide* global sections in $\mathcal{S}(X)$ in which genuine global sections do not exist.

Several examples are discussed in detail in [5]. It is shown that cohomological obstructions over $\mathbb{Z}$ provide witnesses of strong contextuality for a number of well-studied models, including: the GHZ model [12], the Peres–Mermin "magic" square [25, 22], and the 18-vector Kochen–Specker model [7], the PR box [27], and the Specker triangle [29, 16]. These results will be subsumed and greatly generalised in §6.

The coefficients for cohomology can be taken from any commutative ring R. Here is how the cohomological obstructions obtained with different rings relate to each other:

▶ **Proposition 20.** *Let $h\colon R' \longrightarrow R$ be a ring homomorphism. Then, for any $C \in \mathcal{M}$ and $s \in \mathcal{S}(C)$, $\gamma_{F_{R'}\mathcal{S}}(s) = 0$ implies $\gamma_{F_R\mathcal{S}}(s) = 0$, and so $\mathsf{CSC}_R \Rightarrow \mathsf{CSC}_{R'}$ and $\mathsf{CLC}_R \Rightarrow \mathsf{CLC}_{R'}$.*

Proof. If $h\colon R \longrightarrow R'$ is a ring homomorphism, then for any set X there is a map

$$F_h\colon F_RX \longrightarrow F_{R'}X :: r \longmapsto h \circ r \ .$$

which is a group homomorphism. Moreover, this assignment is natural in X. Hence, this determines a presheaf map $F_R\mathcal{S} \longrightarrow F_{R'}\mathcal{S}$, and compatible families on the former are mapped to compatible families on the latter. Since the map F_h above leaves the elements of the generating set fixed, the condition that the family agrees with s_0 at context C_0 is preserved. ◀

We conclude this section with a remark. If $\{r_C \in F_R\mathcal{S}(C)\}_{C\in\mathcal{M}}$ is a compatible family, then the sum of the coefficients of the formal linear combinations r_C is the same for all C. This holds because $\mathcal{S}(\varnothing) = \mathcal{E}(\varnothing) = \{\star\}$; so that for any $C \in \mathcal{M}$, we have

$$r_C|_\varnothing(\star) = \sum_{s\in\mathcal{S}(C)} r_C(s) \ ;$$

i.e. compatibility forces all these restrictions to the empty context to be the same. Therefore, when the obstruction of a section $s_0 \in \mathcal{S}(C_0)$ (more precisely, of the linear combination $1\cdot s_0 \in F_R\mathcal{S}(C_0)$) vanishes, the corresponding family of linear combinations $\{r_C \in F_R\mathcal{S}(C)\}_{C\in\mathcal{M}}$ must in fact contain only *affine* combinations – those whose coefficients sum to one.

6 Cohomology and AvN arguments

The aim of this section is to show that if an empirical model is AvN_R, then the cohomological obstructions witness its strong contextuality. Moreover, it is enough to consider cohomology with coefficients in the ring R itself.

The result is stated as follows.

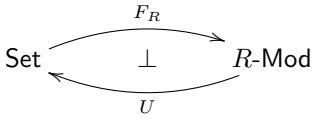
▶ **Theorem 21.** *Let $\mathcal{S}$ be an empirical model on $\langle X, \mathcal{M}, R\rangle$. Then:*

$$\mathsf{AvN}_R(\mathcal{S}) \ \Rightarrow \ \mathsf{SC}(\mathsf{Aff}\,\mathcal{S}) \ \Rightarrow \ \mathsf{CSC}_R(\mathcal{S}) \ \Rightarrow \ \mathsf{CSC}_{\mathbb{Z}}(\mathcal{S}) \ \Rightarrow \ \mathsf{SC}(\mathcal{S}) \ .$$

The first of the implications was already established in Proposition 9, the third in Proposition 20, and the fourth in Proposition 19.

In order to prove the second, we use the properties of the functor $F_R\colon \mathsf{Set} \longrightarrow R\text{-}\mathsf{Mod}$ that constructs the R-module of formal R-linear combinations of elements of a set X. As

already mentioned, $F_R X$ is the free R-module generated by X. This means that it is the left adjoint of the forgetful functor $U\colon R\text{-}\mathsf{Mod} \longrightarrow \mathsf{Set}$.

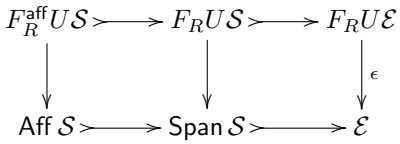


The unit η of this adjunction is the obvious embedding, which we have been using, taking an element $x \in X$ to the formal linear combination $1 \cdot x$. The counit is the natural transformation $\epsilon\colon F_R \circ U \Rightarrow Id_{R\text{-}\mathsf{Mod}}$ given, for each R-module M, by the evaluation map

$$\epsilon_M\colon F_R U(M) \longrightarrow M :: r \longmapsto \sum_{x \in M} r(x)x \ .$$

We are interested in taking formal linear combinations of subsets of elements. Let us fix a module M and a subset $S \subseteq U(M)$. Then the map ϵ_M, by virtue of being an R-module homomorphism, maps the formal linear combinations of elements of S, $F_R(S)$, which coincide with the linear span in $F_R U(M)$ of $\eta[S] = \{1 \cdot s \mid s \in S\}$, to the linear span of S in M, $\mathsf{span}_M\, S$. Moreover, it maps the formal affine combinations $F_R^{\mathsf{aff}}(S) = \mathsf{aff}_{F_R U(M)}\, \eta[S]$ to the affine closure $\mathsf{aff}_M\, S$.

Recall that we are dealing with measurement scenarios whose outcomes are identified with a ring R, hence where $\mathcal{E}(U)$ are themselves R-modules, i.e. $\mathcal{E}\colon \mathcal{P}(X)^{\mathsf{op}} \longrightarrow R\text{-}\mathsf{Mod}$. As such, the counit can be horizontally composed to yield a natural transformation, or map of presheaves, $\mathsf{id}_{\mathcal{E}} * \epsilon\colon F_R \circ U \circ \mathcal{E} \longrightarrow \mathcal{E}$, given at each context $U \subseteq X$ by $\epsilon_{\mathcal{E}(U)}\colon F_R U \mathcal{E}(U) \longrightarrow \mathcal{E}(U)$. Now, given an empirical model $\mathcal{S}$, we can apply the observation regarding subsets of the module at each context. But, since $\mathsf{aff}_{\mathcal{E}(U)}\, \mathcal{S}(U) = (\mathsf{Aff}\, \mathcal{S})(U)$ by definition for U beneath the cover, and since containment still holds above it, we conclude that the presheaf map restricts as follows:



Proof of Theorem 21. We show the contrapositive. Suppose that $\mathcal{S}$ is not CSC_R, i.e. that $\gamma_{F_R\mathcal{S}}(s_0) = 0$ for some $s_0 \in \mathcal{S}(C_0)$. Then, by Proposition 16, this is equivalent to the existence of a compatible family $\{r_C \in F_R\mathcal{S}(C)\}_{C \in \mathcal{M}}$ with $r_{C_0} = s_0$. As observed at the end of §5.3, all these r_C must be formal affine combinations of elements in $\mathcal{S}(C)$. But then the presheaf map $F_R^{\mathsf{aff}} U\mathcal{S} \longrightarrow \mathsf{Aff}\, \mathcal{S}$ above pushes this compatible family to a compatible family of $\mathsf{Aff}\, \mathcal{S}$, implying that the model $\mathsf{Aff}\, \mathcal{S}$ is not strongly contextual. ◀

Essentially the same strategy can be used to prove an analogous result for logical contextuality. The notion of inconsistent theory has to be adapted: instead of asking whether there is a global assignment satisfying all the equations in the theory, we can ask, given a partial assignment $s_0 \in \mathcal{E}(C_0)$ whether there is such a global assignment with the additional requirement that it restricts to s_0. This can be seen as a generalisation of the notion of *robust constraint satisfaction* studied in [4] from the complexity perspective. We write $\mathsf{AvN}_R(e, s_0)$ if the theory of $\mathcal{S}$ has no solution extending s_0. Then we have:

$$\mathsf{AvN}_R(e, s_0) \ \Rightarrow\ \mathsf{LC}(\mathsf{Aff}\, \mathcal{S}, s_0) \ \Rightarrow\ \mathsf{CLC}_R(\mathcal{S}, s_0) \ \Rightarrow\ \mathsf{CLC}_{\mathbb{Z}}(\mathcal{S}, s_0) \ \Rightarrow\ \mathsf{LC}(\mathcal{S}, s_0) \ .$$

Our results show that, where there is an cohomological obstruction, it witnesses genuine contextuality. On the other hand, the important class of AvN examples are all captured by cohomology. It is worth emphasising that all known quantum examples of strong contextuality are of AvN type, hence all such examples are captured by cohomology.

Discussion

We have shown that for a large class of models, their logical or strong contextuality is witnessed by cohomology. This subsumes and greatly generalises the results in [5]. Moreover, these models include a large class of concrete constructions arising from stabiliser quantum mechanics, going well beyond existing results of this kind in the quantum information literature. It remains an objective for future work to achieve a precise characterisation of what cohomology detects, and more generally full equivalences between the various ways of expressing contextuality. Note that, as already mentioned, the first implication in Theorem 21 can be reversed under the assumption that R is a field. If we use a more abstract notion of equational consistency, in terms of quotient modules rather than equations expressed in a "coordinatized" form, then it can be reversed even for general rings. The point of taking the ground ring to be a field is exactly that it allows coordinatization.

We also remark that the cohomological methods we have developed can be applied to an elaborated version of the treatment of logical paradoxes we gave in §2, following the lines of [10, 30]. We aim to give a detailed treatment of this "cohomology of paradox" in future work. We also note the intriguing resemblances, on the conceptual level at least, to the work of Roger Penrose in [24].

Acknowledgements. We thank Alexandru Baltag, Louis Narens and Nicholas Teh for useful discussions. Support from the following is gratefully acknowledged: Templeton World Charity Foundation, AFOSR, EPSRC, the Oxford Martin School, and FCT – Fundação para a Ciência e Tecnologia (Portuguese Foundation for Science and Technology), PhD grant SFRH/BD/94945/2013.

References

1 Samson Abramsky. Relational databases and Bell's theorem. In Val Tannen, Limsoon Wong, Leonid Libkin, Wenfei Fan, Wang-Chiew Tan, and Michael Fourman, editors, *In search of elegance in the theory and practice of computation*, volume 8000 of *LNCS*, pages 13–35. Springer, 2013.

2 Samson Abramsky, Rui Soares Barbosa, Kohei Kishida, Raymond Lal, and Shane Mansfield. Contextuality, cohomology and paradox. arXiv preprint arXiv:1502.03097, 2015.

3 Samson Abramsky and Adam Brandenburger. The sheaf-theoretic structure of non-locality and contextuality. *New J. Phys.*, 13(11):113036, 2011.

4 Samson Abramsky, Georg Gottlob, and Phokion G. Kolaitis. Robust constraint satisfaction and local hidden variables in quantum mechanics. In Francesca Rossi, editor, *Proceedings of the Twenty-Third IJCAI*, pages 440–446. AAAI Press, 2013.

5 Samson Abramsky, Shane Mansfield, and Rui Soares Barbosa. The cohomology of non-locality and contextuality. In Bart Jacobs, Peter Selinger, and Bas Spitters, editors, *Proc. 8th International Workshop on Quantum Physics and Logic 2011*, volume 95 of *EPTCS*, pages 1–14, 2012.

6 John S. Bell. On the Einstein-Podolsky-Rosen paradox. *Physics*, 1(3):195–200, 1964.

7 Adán Cabello, José M. Estebaranz, and Guillermo García-Alcaine. Bell-Kochen-Specker theorem. *Phys. Lett. A*, 212(4):183–187, 1996.

8 Adan Cabello, Simone Severini, and Andreas Winter. (Non-)Contextuality of Physical Theories as an Axiom. *arXiv:1010.2163*, 2010.

9 Carlton Caves. Stabilizer formalism for qubits. Available at `http://info.phys.unm.edu/~caves/reports/stabilizer.ps`, 2006.

10 Roy T Cook. Patterns of paradox. *The Journal of Symbolic Logic*, 69(03):767–774, 2004.

11 Ronald Fagin, Alberto O. Mendelzon, and Jeffrey D. Ullman. A simplified universal relation assumption and its properties. *ACM Transactions on Database Systems (TODS)*, 7(3):343–360, 1982.

12 Daniel M. Greenberger, Michael A. Horne, Abner Shimony, and Anton Zeilinger. Bell's theorem without inequalities. *Am. J. Phys.*, 58(12):1131–1143, 1990.

13 Lucien Hardy. Nonlocality for two particles without inequalities for almost all entangled states. *Phys. Rev. Lett.*, 71(11):1665–1668, 1993.

14 Mark Howard, Joel Wallman, Victor Veitch, and Joseph Emerson. Contextuality supplies the 'magic' for quantum computation. *Nature*, 510(7505):351–355, 06 2014.

15 Raymond Lal. A sheaf-theoretic approach to cluster states, 2011. Private communication.

16 Yeong-Cherng Liang, Robert W. Spekkens, and Howard M. Wiseman. Specker's parable of the overprotective seer. *Phys. Rep.*, 506(1):1–39, 2011.

17 Saunders Mac Lane and Ieke Moerdijk. *Sheaves in geometry and logic*. Springer, 1992.

18 David Maier, Jeffrey D. Ullman, and Moshe Y. Vardi. On the foundations of the universal relation model. *ACM Transactions on Database Systems (TODS)*, 9(2):283–308, 1984.

19 Shane Mansfield. Completeness of Hardy non-locality: Consequences & applications. In *Informal Proceedings of 11th International Workshop on Quantum Physics & Logic*, 2014.

20 N. David Mermin. Extreme quantum entanglement in a superposition of macroscopically distinct states. *Phys. Rev. Lett.*, 65(15):1838–1840, 1990.

21 N. David Mermin. Simple unified form for the major no-hidden-variables theorems. *Phys. Rev. Lett.*, 65(27):3373–3376, 1990.

22 N. David Mermin. Hidden variables and the two theorems of John Bell. *Rev. Mod. Phys.*, 65(3):803–815, 1993.

23 M.Q.C. Nielsen and I. Chuang. *Quantum computation and quantum information*. Cambridge University Press, 2000.

24 Roger Penrose. On the cohomology of impossible figures. *Leonardo*, 25(3/4):245–247, 1992.

25 A. Peres. Incompatible results of quantum measurements. *Phys. Lett. A*, 151(3-4):107–108, 1990.

26 Stefano Pironio, Jean-Daniel Bancal, and Valerio Scarani. Extremal correlations of the tripartite no-signaling polytope. *J. Phys. A–Math. Theor.*, 44(6):065303, 2011.

27 Sandu Popescu and Daniel Rohrlich. Quantum nonlocality as an axiom. *Found. Phys.*, 24(3):379–385, 1994.

28 Robert Raussendorf and Hans J Briegel. A one-way quantum computer. *Phys. Rev. Lett.*, 86(22):5188, 2001.

29 Ernst Specker. Die Logik nicht gleichzeitig entscheidbarer Aussagen. *Dialectica*, 14:239–246, 1960.

30 Michał Walicki. Reference, paradoxes and truth. *Synthese*, 171(1):195–226, 2009.

31 Lan Wen. Semantic paradoxes as equations. *Math. Intell.*, 23(1):43–48, 2001.

32 Xiang Zhang, Mark Um, Junhua Zhang, Shuoming An, Ye Wang, Dong-ling Deng, Chao Shen, Lu-Ming Duan, and Kihwan Kim. State-independent experimental test of quantum contextuality with a single trapped ion. *Phys. Rev. Lett.*, 110(7):070401, 2013.

33 Chong Zu, Y-X Wang, D-L Deng, X-Y Chang, Ke Liu, P-Y Hou, H-X Yang, and L-M Duan. State-independent experimental test of quantum contextuality in an indivisible system. *Phys. Rev. Lett.*, 109(15):150401, 2012.

A Model for Behavioural Properties of Higher-order Programs

Sylvain Salvati[1] and Igor Walukiewicz[2]

1 **Université de Bordeaux, INRIA, LaBRI UMR5800, France**
2 **Université de Bordeaux, CNRS, LaBRI UMR5800, France**

Abstract

We consider simply typed lambda-calculus with fixpoints as a non-interpreted functional programming language: the result of the execution of a program is its normal form that can be seen as a potentially infinite tree of calls to built-in operations. Properties of such trees are properties of executions of programs and monadic second-order logic (MSOL) is well suited to express them.

For a given MSOL property we show how to construct a finitary model recognizing it. In other words, the value of a lambda-term in the model determines if the tree that is the result of the execution of the term satisfies the property. The finiteness of the construction has as consequences many known results about the verification of higher-order programs in this framework.

1998 ACM Subject Classification F.3 Logics and Meaning of Programs

Keywords and phrases Simply typed λY-calculus, Monadic second order logic, semantic models

Digital Object Identifier 10.4230/LIPIcs.CSL.2015.229

1 Introduction

Higher-order functions are being adopted by most mainstream programming languages. Higher-order functions not only increase modularity and elegance of the code, but also help to address such fundamental issues as scalability and fault-tolerance. In consequence, higher-order functions are increasingly used for writing programs interacting with an environment, like, for example, client-server web applications. To accompany this evolution, new kinds of analysis tools are needed, focusing on behavioural properties of higher-order functional programs. For example, some guidelines for secure web programming may require that if a database access is required infinitely often then calls to a logging function must be made again and again. Our objective is to develop denotational models for such kinds of properties.

We consider λY-calculus, the simply typed λ-calculus with fixpoints, as an abstraction of a higher-order programming language that faithfully represents the control flow. Under the name of recursive program schemes the calculus has been studied since 1960s [11, 5, 6, 7, 13, 19]. The particularity of this approach is to focus on the free interpretation: all constants are non-interpreted symbols and the interpretation of a term is a tree composed from constants. In the context of λ-calculus this tree is called the Böhm tree. Figure 1 presents the Böhm tree of the `map` function. It is a generic iterator taking a function and a list, and applying the function to every element of the list. Observe that even for such a simple program its Böhm tree is infinite and not regular.

Program properties can be grouped in two families. The first, and the most obvious one, concerns the absence of run-time errors. A slogan "well-typed programs never go wrong" clearly expresses this idea. More generally, this family contains all kinds of *safety properties*, i.e., those determined by a set of finite executions considered as admissible. The other family is that of *liveness properties* that talk about infinite executions. For example: "logging

24th EACSL Annual Conference on Computer Science Logic (CSL 2015).
Editor: Stephan Kreutzer; pp. 229–243

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

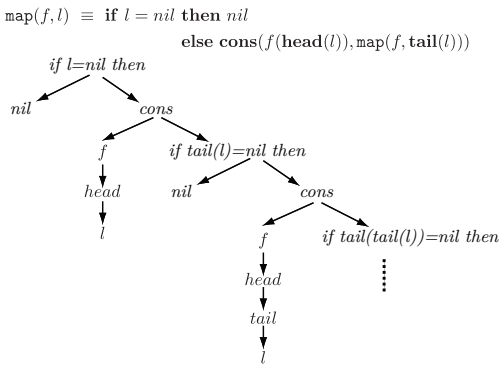


Figure 1 The map function and its semantics in the form of a (simplified) Böhm tree.

function is called again and again" or "every initiated communication is eventually closed". Concerning the `map` function, we can say for example that if l is a finite list then the call $\texttt{map}(f, l)$ evokes f only finitely many times. In fact all fairness properties are particular liveness properties. Such properties are of relevance to servers, web services, operating systems, and more generally, to all kinds of interactive applications. Regarding liveness properties, monadic second-order logic (MSOL), or equivalently automata on infinite objects, sets the standard of expressivity and algorithmic manageability. Moreover, thanks to the result of Ong [19], it is decidable if the Böhm tree of a given term of the λY-calculus satisfies a given MSOL property.

In this paper we show how to construct for a given MSOL property a finitary model so that the value of a term in the model determines if the Böhm tree of the term satisfies the property. More precisely, we work with the formalism of parity automata instead of MSOL. We show that the value of a term of the base type in the model constructed from a given automaton is simply the set of states from which the automaton accepts the Böhm tree of the term (Theorem 12).

Our model construction shows how to extend Scott models to integrate the parity condition of a given automaton. Finitary Scott models are the simplest models of the λY-calculus: the base type is interpreted as a finite lattice, functional types as the sets of monotone functions, and the fixpoint as the least fixpoint. Such models correspond in a precise sense to safety properties, or equivalently to finite automata on infinite trees that are Ω-blind and have trivial acceptance conditions [22]. This implies that in order to capture the expressive power of parity automata some modification of Scott models is needed. The straightforward idea of introducing ranks of the parity condition directly in the base type does not seem to work. Instead our construction introduces ranks only in higher types. The other crucial point is the interpretation of function spaces: we cannot take all monotone functions but only those that behave well with respect to ranks. This is formalized with a new domain identity we call *stratification*.

The model construction gives a completely compositional approach to verification: the result of a term is calculated from the results for its subterms. In particular, we give the meaning of a fixpoint constant as a particular fixpoint of its argument. The construction implies the transfer theorem for MSOL [21], and with it a number of consequences offered by this theorem. Finitary models are used in program transformations: during its execution the program can calculate the values of chosen subterms [22, 9]. In our case it can, for example, detect if an argument satisfies a particular liveness property.

Our construction is based on the insights from a very influential paper of Kobayashi and Ong [15], where, amongst other contributions, they give a type system to capture the

same dependencies inside terms that we represent in our model. Although the quest for models for behavioural properties has begun some time ago, the results started to appear only recently. Tsukada and Ong [27] extended the approach from [15] to a type system for the whole λY-calculus. In this system the fixpoint is still treated externally via games, and the model underlying the system is not finitary. They use game semantics to understand a difficult problem of the behaviour of the application operation at the level of Böhm trees. Also last year, Hofmann and Chen provided a model for verifying path properties expressed in MSOL [10]. Their construction is restricted only to first-order λY-terms. They use in an elegant way Wilke algebras that are an algebraic notion of recognizer for languages of infinite words. One of the problems we are facing here is that there does not exist equally satisfying notion of an algebraic recognizer for infinite trees. Even if we wanted to stay with properties of paths, it is not clear how to extend Wilke algebras to higher orders, the problem being to find an admissible class of fixpoint operations. More recently, Grellois and Mellies [8] have given a categorical account of the behaviour of ranks in a model. They derive an infinite model via elegant general constructions. About the same time, we have provided a model construction for properties expressed in weak MSOL [25]. The model is a sort of layered Scott model. The restriction to weak MSOL greatly simplifies the integration of ranks in the model. As a consequence, it was possible to adapt classical arguments from domain theory to prove the correctness of the model. The present construction does not follow the line of [25]. Apart from [15], the main influence comes from the work of Mellies [17] clearly showing the value of using the morphism composition similar to that in Kleisli categories. Furthermore, the stratification property is essential to get the model to satisfy the required equations. The proof methods for the correctness of the model are extensions of game based methods we have developed for the proof of the transfer theorem [21]. With respect to other proposals [15, 27, 8] (which capture the so-called Ω-blind automata [22]) our model is capturing automata that are able to detect the divergence symbol Ω: for a run to be accepting, the ranks assigned to the nodes labelled Ω should be even. On a model side, stratification condition is essential.

Apart from model based approaches cited above, there is a very active research in verification of behavioural properties of higher-order programs. Among the closest methods using the class of properties and programs we consider here we can list [14, 4, 20]. Similar research objectives are also pursued in different settings. We would like to mention the work of Naik and Palsberg [18] who make a connection between model-checking and typing. They consider only safety properties, and focus on first-order imperative programs. Another interesting line of research is proposed by Jeffrey [12] who shows how to incorporate Linear Temporal Logic into types using a richer dependent types paradigm. The calculus is intended to talk about control and data in functional reactive programming framework, and aims at using SMT solvers.

Organization of the paper: In the next preliminary section we introduce basic definitions, and present two special cases that allow us to introduce the main concepts in a simpler setting. Section 3 is devoted to the definition of the model and its properties. The main theorem of the paper is stated in this section. Section 4 shows some consequences of the model construction. The conclusions section outlines some directions for further research. A long version of the paper that gives the details of the proofs is available [24].

2 Preliminaries

We start by introducing λY-calculus and parity automata. Then we present two simple special cases of the main result of the paper. The first case shows what can be achieved with the classical notion of a model for λY-calculus. The second considers only terms of order at most 1. It allows us to introduce some crucial elements of the general solution.

2.1 λY-calculus

The *set of types* is constructed from a unique *basic type* o using a binary operation $\to$ that associates to the right. Thus o is a type and if A, B are types, so is $(A \to B)$. The order of a type is defined by: $\mathit{order}(o) = 0$, and $\mathit{order}(A \to B) = \mathit{max}(1 + \mathit{order}(A), \mathit{order}(B))$. We work with *tree signatures* that are finite sets of *typed constants of order at most* 1. Types of order 1 are of the form $o \to \cdots \to o \to o$ that we abbreviate $o^i \to o$ when they contain $i+1$ occurrences of o. For convenience we assume that $o^0 \to o$ is just o. If Σ is a signature, we write $\Sigma^{(i)}$ for the set of constants of type $o^i \to o$.

Simply typed λY-terms are built from the constants in the signature, and constants Y^A, Ω^A for every type A. These stand for the *fixpoint combinator* and *undefined term* and they respectively have type $(A \to A) \to A$ and A. Apart from constants, for each type A there is a countable set of variables $x^A, y^A, \ldots$. Terms are built from these constants and variables using typed application and λ-abstraction. We shall write sequences of λ-abstractions $\lambda x_1 \ldots \lambda x_n.\ M$ with only one λ: either $\lambda x_1 \ldots x_n.\ M$, or even shorter $\lambda \vec{x}.\ M$. The usual operational semantics of the λ-calculus is given by β-reduction and δ-reduction. The corresponding contraction rules are $(\lambda x.M)N \to_\beta M[N/x]$ and $YM \to_\delta M(YM)$.

The *Böhm tree* of a term M is obtained by reducing it until one reaches a term of the form $\lambda \vec{x}.N_0 N_1 \ldots N_k$ with N_0 a variable or a constant. Then $BT(M)$ is a tree having its root labelled by $\lambda \vec{x}.N_0$ and having $BT(N_1), \ldots, BT(N_k)$ as subtrees. Otherwise $BT(M) = \Omega^A$, where A is the type of M. Böhm trees are infinite normal forms of λY-terms. A Böhm tree of a closed term of type o over a tree signature is a potentially infinite ranked tree: a node labelled by a constant a of type $o^i \to o$ has i successors. Among constants Ω^A, only constant Ω^o can appear in the Böhm tree of such a term.

2.2 MSOL and parity automata

We are interested in properties of trees expressed in monadic second-order logic (MSOL). This is an extension of first-order logic with quantification over sets of elements. Over infinite trees MSOL formulas define precisely regular tree languages. This class of languages has numerous other characterizations. Here we will rely on the one using parity tree automata.

Automata will work on Σ-labelled trees, where Σ is a tree signature. Trees are partial functions $t : \mathbb{N}^* \dot{\to} \Sigma \cup \{\Omega\}$ such that the number of successors of a node is determined by the label of the node. In particular, if $t(u) \in \Sigma^{(0)}$ then u is a leaf. The *nodes* of t, are the elements of the domain of t. The set of nodes should be prefix closed. A *label* of a node u is $t(u)$.

We will use nondeterministic max-parity automata, that we will call *parity automata* for short. Such an automaton accepts trees over a fixed tree signature Σ. It is a tuple

$$\mathcal{A} = \langle Q, \Sigma, \{\delta_i\}_{i \in \mathbb{N}}, rk : Q \to [m] \rangle$$

where Q is a finite set of states, rk is the *rank function* with the range $[m] = \{0, \ldots, m\}$, and $\delta_i : Q \times \Sigma^{(i)} \to \mathcal{P}(Q^i)$ is the *transition function*. Observe that since the signature Σ is

finite, only finitely many δ_i are nontrivial. From the definition it follows that, for example, $\delta_2 : Q \times \Sigma^{(2)} \to \mathcal{P}(Q \times Q)$ and $\delta_0 : Q \times \Sigma^{(0)} \to \{\emptyset, \{\emptyset\}\}$. We will simply write δ without a subscript when this causes no ambiguity. We require that $\delta(q, \Omega^o) = \{\emptyset\}$ if the rank of q is even, and $\delta(q, \Omega^o) = \emptyset$ otherwise[1].

A *run of* $\mathcal{A}$ *on* t *from a state* q^0 is a labelling of nodes of t with the states of $\mathcal{A}$ such that: (i) the root is labelled with q^0, (ii) if a node u is labelled q and its k-successors (with $k > 0$) are labelled by $q_1, \dots q_k$, respectively, then $(q_1, \dots, q_k) \in \delta_k(q, t(u))$; recall that $t(u)$ is the letter labelling the node u.

A run is *accepting* when: (i) for every leaf u of t, if q is the state of the run in u then $\delta_0(q, t(u)) = \{\emptyset\}$, and moreover (ii) for every infinite path of t, the labelling of the path given by the run satisfies the *parity condition*. This means that if we look at the ranks of states assigned to the nodes of the path then the maximal rank appearing infinitely often is even. A tree is *accepted by* $\mathcal{A}$ *from a state* q^0 if there is an accepting run from q^0 on the tree.

It is well known that for every MSOL formula there is a parity automaton recognizing the set of trees that are models of the formula. The converse also holds. Let us also recall that the automata model can be extended to alternating parity automata without increasing the expressive power. Here, for simplicity of the presentation, we will work only with nondeterministic automata but our constructions apply also to alternating automata.

In the context of verification of higher-order properties, automata with *trivial acceptance conditions* have gathered considerable attention [14]. These are obtained by requiring that all states have rank 0. In terms of runs it means that every run of such an automaton on an infinite tree without leaves is accepting. For the reasons that will be apparent in the next subsection one more simplifying condition is imposed in the literature. An automaton is Ω-*blind* if $\delta(q, \Omega) = \{\emptyset\}$ for all states q. So Ω-blind automaton unconditionally accepts divergent computations, while our definition allows to test divergence with the rank of the state.

A parity automaton together with a state *recognizes* a language of closed terms of type o:

$$L(\mathcal{A}, q^0) = \{M : M \text{ is closed term of type } o,\ BT(M) \text{ is accepted by } \mathcal{A} \text{ from } q^0\} .$$

2.3 Models with the least fixpoint

A Scott model associates to each type A a finite lattice $\mathcal{D}_A$ in which λY-terms of type A can be interpreted. For a type $B \to C$, this lattice is the set of monotone functions f from $\mathcal{D}_B$ to $\mathcal{D}_C$. The set $\mathcal{D}_{B \to C}$ is ordered pointwise ($f \leq g$ when for every $b \in \mathcal{D}_B$, $f(b) \leq g(b)$) making it a lattice. Constants are interpreted as functions of the appropriate type. Fixpoint operators Y are interpreted as the least fixpoints.

The semantics of a term M of type A in a given valuation υ, denoted $[\![M, \upsilon]\!]$, is an element of $\mathcal{D}_A$. As usual, a valuation is a partial function from variables to elements of the model respecting types: if defined $\upsilon(x^A)$ is an element of $\mathcal{D}_A$. We use $\emptyset$ for the empty valuation. The inductive definition of the semantics is presented in Figure 2. For illustration we have also included a clause for constants and implicitly assumed that $\mathcal{D}_o$ is of the form $\mathcal{P}(Q)$. It explains the case when we would like to construct a model from an automaton as stated in the theorem below. Let us remark that Theorem 1 allows to make a boolean combination Ω-blind automata when constants have arbitrary interpretation in arbitrary finitary Scott model.

[1] This unusual treatment of Ω^o is a small but important ingredient of our construction. Any other choice looses the correspondance between ranks in $\mathcal{A}$ and fixpoint alternation in the definition of the fixpoint.

$$\begin{aligned} [\![x, v]\!] &= v(x) & [\![a, v]\!]h_1 \dots h_k &= \{q : \exists_{(q_1,\dots,q_k)\in\delta(q,a)} q_i \in h_i \text{ for all } i\} \\ [\![\lambda x.M, v]\!]h &= [\![M, v[h/x]]\!] & [\![MN, v]\!] &= [\![M, v]\!]([\![N, v]\!]) \\ [\![Y]\!]\, f &= \bigvee\{f^n(\bot) \mid n \in \mathbb{N}\} & [\![\Omega]\!] &= \bot \end{aligned}$$

Figure 2 Semantics in a Scott model.

A Scott model can be used to recognize a set of terms. A subset R of $\mathcal{D}_o$ is said to *recognize* the set of closed λY-terms M of type o whose semantics is in R, i.e. $[\![M, \emptyset]\!] \in R$. This notion of recognition [23] generalizes the usual notion of recognition for words by finite monoids. In this way, finitary Scott models determine a class of languages of λY-terms they recognize. The following theorem characterizes this class (that Scott domains capture Ω-blind automata was first established in [1]).

▶ **Theorem 1** ([22]). *A language of λY-terms is recognized by a boolean combination of Ω-blind automata with trivial acceptance condition iff it is recognized by a Scott model where Y constants are interpreted as the least fixpoint.*

This theorem determines the limits of Scott models with least fixpoints. By duality this also applies to models with greatest fixpoints. So in order to capture more properties we need to be able to construct some other fixpoints.

2.4 The case of terms of order at most 1

The case of Scott models clearly pointed out the challenge of a model construction for all parity automata. In this section we will present the special case of our construction for terms of order at most 1. Such terms have only variables of type o and all their subterms are of type of order at most 1. We will construct a model for an arbitrary parity automaton. The advantage of terms of order at most 1 is that we can describe in a direct way what our semantics expresses. The semantic equations for the general case will be the same as here. We hope that this presentation will give some general intuitions about what properties of Böhm trees the model captures, as well as specific intuitions about the operation $(\cdot)\lfloor_r$ (cf. Definition 3) that deals with parity acceptance conditions at the level of semantics. One can see the construction below as a reformulation of the type system of Kobayashi and Ong [15] in terms of models.

For the rest of the subsection we fix a parity automaton $\mathcal{A} = \langle Q, \Sigma, \delta, rk : Q \to [m]\rangle$.

Let us first consider terms without fixpoints. If M is a closed term of type o then $BT(M)$ is a finite tree with internal nodes labelled by constants of types of order 1 and leaves labelled by constants of type o. It is clear what is an accepting run of automaton $\mathcal{A}$ on $BT(M)$.

Suppose now that M has free variables, that are necessarily of type o. If M is of type o then $BT(M)$ is still a finite tree but it may have nodes labelled by variables. We can thus consider variables as holes where we can put states and ask whether there is a run. The parity condition requires to keep more information. So in addition to states, we keep track of the maximal ranks of states that appear on the paths from the root to the leaves labelled with variables. This idea is formalized in the following definition and illustrated in Figure 3.

▶ **Definition 2.** Let $M : o$ be a term of order at most 1. Let v be a function assigning to every free variable of M a value from $\mathcal{P}(Q \times [m])$. We say that $\mathcal{A}$ *accepts* $BT(M)$ *from* q *to* v iff there is a run of $\mathcal{A}$ on $BT(M)$ starting in q, satisfying the conditions of an accepting

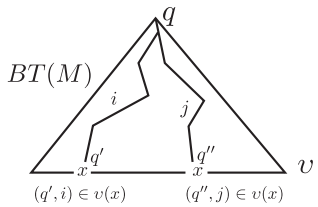


Figure 3 $(q', i), (q'', j) \in \upsilon(x)$ as the maximal color seen on a path from the root to occurrences of x respectively labeled with states q' and q'' are i and j.

run from page 233, and such that for every variable x and leaf of $BT(M)$ labelled by x: if q' is a state of the run in the leaf, and i is the maximal rank of states on the path from the root to the leaf then $(q', i) \in \upsilon(x)$.

We will define a semantics of λ-terms that captures this notion of acceptance. First we define semantic domains for types of order at most 1:

$$\mathcal{D}_o = \mathcal{P}(Q) \qquad \mathcal{R}_o = \mathcal{P}(\{(q, r) : q \in Q \text{ and } rk(q) \leq r \leq m\})$$
$$\mathcal{D}_{o \to \cdots \to o \to o} = \mathcal{R}_o \to \cdots \to \mathcal{R}_o \to \mathcal{D}_o$$

So $\mathcal{R}_o$ is the set of sets of ranked states, with the restriction that the rank should be at least as big as the rank assigned to the state in the automaton. The intended meaning of ranks given by the above definition clearly justifies this restriction. We call the elements of $\mathcal{R}_o$ *residuals*.

Both $\mathcal{D}_o$ and $\mathcal{R}_o$ are ordered by inclusion, and $\mathcal{D}_{o^i \to o}$ is ordered pointwise.

We now introduce the operation $(\cdot)\downarrow_r$ that is handling the parity condition at the level of semantics. Even though the definition may at first sight seem technical, Lemma 4 provides some rather clear intuitions about how it works.

▶ **Definition 3.** For $h \in \mathcal{R}_o$, and $r \in [m]$ we put

$$h\downarrow_r = \{(q, i) \in h : r \leq i\} \cup \{(q, j) : (q, r) \in h, rk(q) \leq j \leq r\} .$$

As an example, observe that $h\downarrow_0 = h$.

▶ **Lemma 4.** *For $h \in \mathcal{R}_o$, $q \in Q$, and $r, r_1, r_2 \in [m]$:*

- $(h\downarrow_{r_1})\downarrow_{r_2} = h\downarrow_{\max(r_1, r_2)}$;
- $(q, rk(q)) \in h\downarrow_r$ *iff* $(q, \max(r, rk(q))) \in h$

The above two properties characterize the family of operations $(\cdot)\downarrow_r$. So Definition 3 is imposed on us if we want to have properties listed in the lemma.

The proof of Proposition 6 below, illustrates how we use the two properties from Lemma 4 to capture in a compositional way the acceptance of Böhm trees of Definition 2.

We also use two other operations. The first is a lifting of elements from $\mathcal{D}_o$ to $\mathcal{R}_o$. The second projects an element of $\mathcal{R}_o$ to $\mathcal{D}_o$ by taking a sort of diagonal.

$$f \cdot r = \{(q, r) : q \in f \text{ and } rk(q) \leq r\} \qquad \text{for } f \in \mathcal{D}_o \text{ and } r \in [m]$$
$$h^\partial = \{q : (q, rk(q)) \in h\} .$$

Given a valuation $\upsilon : Vars \to \mathcal{R}_o$ the semantics of a term M of type A is an element $[\![M, \upsilon]\!] \in \mathcal{D}_A$. Its definition is presented in Figure 4. Put next to the semantics in a Scott model from Figure 2, one can clearly see the differences that are due to the presence of ranks.

$$
\begin{aligned}
[\![x, \upsilon]\!] &= (\upsilon(x))^{\partial} \\
[\![a, \upsilon]\!] h_1 \dots h_k &= \{q : \exists_{(q_1,\dots,q_k)\in\delta(q,a)}\ q_i \in (h_i\restriction_{rk(q)})^{\partial} \text{ for all } i\} \\
[\![\lambda x.M, \upsilon]\!] h &= [\![M, \upsilon[h/x]]\!] \\
[\![MN, \upsilon]\!] &= [\![M, \upsilon]\!] \langle\!\langle N, \upsilon \rangle\!\rangle \quad \text{where } \langle\!\langle N, \upsilon \rangle\!\rangle = \bigvee_{r=0}^{m} \left([\![N, \upsilon\restriction_r]\!] \cdot r \right)
\end{aligned}
$$

Figure 4 Semantics in an extension of the Scott model with ranks.

For example, in the variable rule it is necessary to convert the meaning of a variable from $\mathcal{R}_o$ to $\mathcal{D}_o$. Later, in the application rule, it is necessary to lift the meaning of N from $\mathcal{D}_o$ to $\mathcal{R}_o$. The notation $\upsilon\restriction_r$ means υ where $(.)\restriction_r$ is applied pointwise.

In our characterization of the semantics we will use *step functions*. For $f_1, \dots, f_k \in \mathcal{R}_o$ and $q \in \mathcal{D}_o$ we write

$$f_1 \mapsto \dots \mapsto f_k \mapsto q$$

for the function h of type $\mathcal{R}_o^k \to \mathcal{D}_o$ such that $h(f_1', \dots, f_k') = \{q\}$ if $f_i' \geq f_i$ for all $i = 1, \dots, k$ and $h(f_1', \dots, f_k') = \emptyset$ otherwise. A step function $f_1 \mapsto \dots \mapsto f_k \mapsto (q, i)$ for some $(q, i) \in \mathcal{R}_o$ is defined similarly.

▶ **Example 5.** Take a signature with three constants a, b, c of arity $2, 1, 0$, respectively. Consider a parity automaton $\mathcal{A} = \langle \{q_0, q_1\}, \Sigma, \delta, rk : Q \to [1] \rangle$ where the rank of a state is given by its index, and the only pairs for which the value of δ is not $\emptyset$ are $\delta(q_0, a) = Q \times Q$, $\delta(q_1, b) = Q$, and $\delta(q_0, b) = \delta(q_1, c) = \{\emptyset\}$. So from q_0 the automaton recognizes the set of trees with root labelled a and only finitely many b's on every path.

We are going to evaluate the term $a\, x\, (f (b\, x))$ in the model induced by $\mathcal{A}$ and in the valuation υ that maps x to $\{(q_1, 1)\}$ and f to the step function $\{(q_1, 1)\} \mapsto (q_0, 0)$. The variable f is meant to represent a closed term so as to make the example not too long. We get $[\![a\, x (f (b\, x)), \upsilon]\!] = \{q_0\}$ with the following calculation:

$$
\begin{aligned}
[\![x, \upsilon]\!] &= \{q_1\} & \langle\!\langle x, \upsilon \rangle\!\rangle &= \{(q_1, 1)\} \\
[\![b\, x, \upsilon]\!] &= \{q_1\} & \langle\!\langle b\, x, \upsilon \rangle\!\rangle &= \{(q_1, 1)\} \\
[\![f (b\, x), \upsilon]\!] &= \{q_0\} & \langle\!\langle f (b\, x), \upsilon \rangle\!\rangle &= \{(q_0, 0)\} \\
[\![a\, x (f (b\, x)), \upsilon]\!] &= \{q_0\}\ .
\end{aligned}
$$

▶ **Proposition 6.** *$[\![M, \upsilon]\!] \geq f_1 \mapsto \dots \mapsto f_k \mapsto q$ iff for some fresh variables $z_1 \dots z_k$, $\mathcal{A}$ accepts $BT(M z_1 \dots z_k)$ from q to $\upsilon[f_1/z_1 \dots f_k/z_k]$.*

Proof. The case of a variable follows by unrolling the definitions. If $BT(M)$ is just the variable, $\mathcal{A}$ accepts $BT(M)$ from q to υ iff $(q, rk(q)) \in \upsilon$. This is because the maximal rank of a state seen from the root of $BT(M)$ to the leaf (which are the same nodes) is $rk(q)$.

A more interesting case is that of a constant a, say it is of a type $o \to o \to o$. For the left to right implication, suppose $[\![a, \upsilon]\!] \geq f_1 \to f_2 \to q$. We need to show that $a z_1 z_2$ admits a run from q to a valuation $\upsilon[f_1/z_1, f_2/z_2]$. From the definition of the semantics we have $(q_1, q_2) \in \delta(q, a)$ such that $(q_i, rk(q_i)) \in f_i\restriction_{rk(q)}$. By Lemma 4 we get $(q_i, \max(rk(q_i), rk(q))) \in f_i$. So we can take a run on $a z_1 z_2$ assigning q to the root and q_1, q_2 to the leafs labelled z_1, z_2, respectively. Since indeed $\max(rk(q_i), rk(q))$ is the maximal rank seen in the run from the

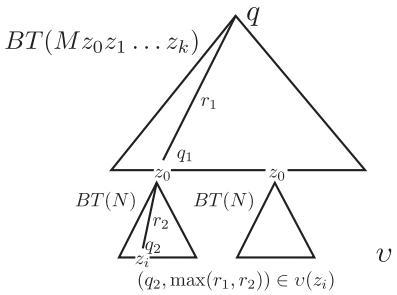


Figure 5 The case of application.

root to z_i this shows that $\mathcal{A}$ accepts az_1z_2 from q to υ. The other direction is analogous thanks to the equivalence in Lemma 4.

We consider the case of the application. We will only present the left to right direction. Suppose $[\![MN, \upsilon]\!] \geq f_1 \mapsto \cdots \mapsto f_k \mapsto q$, and let us look what is the semantics of the application. Since we are considering only terms of order at most 1, N is of type o and $\langle\!\langle N, \upsilon \rangle\!\rangle$ is in $\mathcal{R}_o$. We have $[\![M, \upsilon]\!]\langle\!\langle N, \upsilon \rangle\!\rangle \geq f_1 \mapsto \cdots \mapsto f_k \mapsto q$, which is the same as $[\![M, \upsilon]\!] \geq \langle\!\langle N, \upsilon \rangle\!\rangle \mapsto f_1 \mapsto \cdots \mapsto f_k \mapsto q$. Now the induction hypothesis tells us that $BT(Mz_0z_1\dots z_k)$ is accepted by $\mathcal{A}$ from q to $\upsilon[\langle\!\langle N, \upsilon \rangle\!\rangle/z_0, f_1/z_1, \dots f_k/z_k]$. Now let us look what it means that $(q', r') \in \langle\!\langle N, \upsilon \rangle\!\rangle$. By unfolding the definitions we obtain $q' \in [\![N, \upsilon\restriction_{r'}]\!]$. Using the induction hypothesis for N, we have a run of $\mathcal{A}$ on $BT(N)$ from q' to $\upsilon\restriction_{r'}$. From these observations we construct a required run on $BT(MNz_1\dots z_k)$ from q to υ.

Observe that $BT(MNz_1\dots z_k)$ is obtained from $BT(Mz_0z_1\dots z_k)$ by plugging in every leaf labelled z_0 the tree $BT(N)$ (cf. Figure 5). We want to construct on $BT(MNz_1\dots z_k)$ a run from q to υ. For this we just take a run on $BT(Mz_0z_1\dots z_k)$ from q to the valuation $\upsilon[\langle\!\langle N, \upsilon \rangle\!\rangle/z_0, f_1/z_1, \dots f_k/z_k]$. Then for every leaf l of $BT(Mz_1\dots z_k)$ labelled z_0 with q_l the state of the run in l and r_l the maximal rank from the root to l, we prolong the run with the run on $BT(N)$ from q_l to $\upsilon\restriction_{r_l}$.

To show that this run is as required we take a leaf l_2 of $BT(MNz_1\dots z_k)$ labelled by some variable y. We suppose that q_2 is the state assigned by the run to l_2 and that r is the maximal rank of states of the run on the path from the root to l_2. We want to show that $(q_2, r) \in \upsilon(y)$. If l_2 is a leaf of $BT(Mz_0z_1\dots z_k)$ then this directly follows from the definition of the run. If it is not, then the path to l_2 passes through the leaf l_1 of $BT(Mz_0z_1\dots z_k)$ labelled by z_0 and then gets to $BT(N)$; cf. Figure 5. Let q_1 be the state labelling l_1, let r_1 be the maximal rank from the root to l_1, and let r_2 be the maximal rank from l_1 to l_2. By looking at the part of the run on $BT(N)$ we get $(q_2, r_2) \in \upsilon\restriction_{r_1}(y)$. Lemma 4 then gives $(q_2, \max(r_1, r_2)) \in \upsilon(y)$, that is exactly the required property. ◀

The above proof is so simple because the composition of Böhm trees of terms of order at most 1 is easy. We can now try to add a fixpoint to our syntax. We consider terms of the form YM with M of type $o \to o$. The semantics of a term $[\![M, \upsilon]\!]$ is a function from $\mathcal{R}_o$ to $\mathcal{D}_o$. If we want to calculate the semantics of YM then we need to do some manipulation with the function $[\![M, \upsilon]\!]$ as its domain and co-domain are different. The situation becomes clearer when we recall that $\mathcal{R}_o \subseteq \mathcal{P}(Q \times [m])$. So $[\![M, \upsilon]\!]$ is essentially a function of m arguments. This is very fortunate as we can expect that the computation of the semantics of YM needs m fixpoints alternating between the least and the greatest fixpoints.

We will give a general formula for calculating the fixpoint in Section 3 when we fully describe our model. Since we have Y in the syntax, this formula itself should denote an

element of our model. Here let us show the formula for the case of $m = 1$. This means that we have two ranks 0 and 1. Using $f : \mathcal{R}_o \to \mathcal{D}_o$ to denote the function $[\![M, v]\!]$ the semantics $[\![Y\,M, v]\!]$ is given by $F_0 \in \mathcal{D}_o$ defined by

$$F_0 = \nu Z_0.\ f^{\partial}(Z_0 \cdot 0 \cup F_1 \cdot 1) \qquad\qquad F_1 = \mu Z_1 \nu Z_0.\ (f\restriction_1)^{\partial}(Z_0 \cdot 0 \cup Z_1 \cdot 1)$$

We omit a, not so short, proof of the correctness of this formula. The proof for the general case is presented in [24]. The set F_0 is the set of states in which the term M is accepted when it is in a context where the maximal color from the root to it is 0 (this includes the empty context), while F_1 is the set of states in which the term M is accepted when the color is 1. This distinction is only important for terms with free variables, where, as we have seen, the values associated to variables by valuations depend on the context. So for closed terms F_0 and F_1 are equal.

3 A model recognizing MSOL properties

We now extend the definitions we have given in the previous section to higher orders. Mellies [16] sketched a definition of fixpoint that only worked for closed terms. We here give a definition of higher-order fixpoints that work for open terms. As ranks in the model are used to keep track of the context where variables occur, most of the technical difficulties of the construction of the model appear in this definition. With this definition, we obtain a model of the λY-calculus that recognizes terms whose Böhm trees are accepted by a given parity automaton. More precisely, for every closed λY-term M of type o we will have:

$$[\![M, \emptyset]\!] = \{q : \mathcal{A} \text{ accepts } BT(M) \text{ from } q\}\ .$$

For the rest of this section we fix a parity automaton $\mathcal{A} = \langle Q, S, \delta, rk : Q \to [m]\rangle$. In particular, m is the maximal rank of a state of $\mathcal{A}$.

We start by generalizing the definition of residuals $\mathcal{R}_o$ to all types. At the same time we will generalize the operation $(\cdot)\restriction_r$, as well as define a new operation $(\cdot)\Downarrow_q$. For a residual f in $\mathcal{R}_o$, we let $f\Downarrow_q$ be $\{r : (q, r) \in f\}$. Now we define $\mathcal{R}_{A\to B}$ to be the set of monotone functions f that satisfy the following *stratification* property:

$$\forall g \in \mathcal{R}_A.\ \forall q \in Q.\ (f(g))\Downarrow_q = (f(g\restriction_{rk(q)}))\Downarrow_q \qquad (\texttt{strat})$$

at the same time we define for every $g \in \mathcal{R}_A$:

$$f\Downarrow_q(g) = (f(g))\Downarrow_q, \qquad f\restriction_r(g) = (f(g))\restriction_r\ .$$

The elements of $\mathcal{R}_A$ are ordered using the pointwise order. It can be shown that this order makes $\mathcal{R}_A$ a lattice.

For an intuition behind the (`strat`) property it may be useful to look back at Figure 5. Suppose f is the meaning of M and g is the meaning of N. The formula $f(g)\Downarrow_q$ then means that we are interested in the runs on $BT(MN)$ starting from q. As can be seen from the proof of Proposition 6, in such a run every appearance of $BT(N)$ will be lifted with $\restriction_r$ operation where r is the maximal rank seen from the root to this appearance. We do not know what this r will be, but it will be at least $rk(q)$, so it is safe to already apply $\restriction_{rk(q)}$ to g. In other words, for the runs starting in q we should get the same result from $f(g)$ as from $f(g\restriction_{rk(q)})$. Yet another more formal intuition comes from the application clause. The meaning of $\langle\!\langle N, v\rangle\!\rangle$ as a function of v satisfies the (`strat`) property.

As in the previous section, we do not interpret λY-terms in the lattices $\mathcal{R}_A$, but rather in the lattices $\mathcal{D}_A$ that are generalizations at every type of $\mathcal{D}_o$. For this we must define $f\Downarrow_q$ for $f \in \mathcal{D}_A$: we put $f\Downarrow_q = f \cap \{q\}$ for $f \in \mathcal{D}_o$; and $f\Downarrow_q(g) = (f(g))\Downarrow_q$ for $f \in \mathcal{D}_{A\to B}$, and $g \in \mathcal{R}_A$. Using the same notation for the operation $\Downarrow_q$ when it acts on $\mathcal{D}_A$ or $\mathcal{R}_A$ should not confuse the reader as in both cases, it corresponds to focusing on the behaviour of the function on the state q. With this definition we let $\mathcal{D}_{A\to B}$ be the set of monotone functions from $\mathcal{R}_A$ to $\mathcal{D}_B$ that satisfy the same (`strat`) identity.

▶ Remark. The definitions of $(\cdot)\lfloor_r$ and $(\cdot)\Downarrow_q$ are covariant and they become more intuitive when we consider types written as $A_1 \to \cdots \to A_k \to o$, or in an abbreviated form as $\vec{A} \to o$. In this case, using $\to_{ms}$ for the set of monotone and stratified functions, we have:

$$\mathcal{D}_{\vec{A}\to o} = \mathcal{R}_{A_1} \to_{ms} \cdots \to_{ms} \mathcal{R}_{A_k} \to_{ms} \mathcal{D}_o$$
$$\mathcal{R}_{\vec{A}\to o} = \mathcal{R}_{A_1} \to_{ms} \cdots \to_{ms} \mathcal{R}_{A_k} \to_{ms} \mathcal{R}_o$$
$$g\Downarrow_q(\vec{h}) = (g(\vec{h}))\Downarrow_q \qquad g\lfloor_r(\vec{h}) = (g(\vec{h}))\lfloor_r$$

where $\vec{h}$ is a vector of elements from $\mathcal{R}_{A_1} \times \cdots \times \mathcal{R}_{A_k}$, and the operations $\Downarrow_q$, $\lfloor_r$ are applied only to elements from $\mathcal{D}_o$ or $\mathcal{R}_o$, depending on whether g is from $\mathcal{D}_{\vec{A}\to o}$ or $\mathcal{R}_{\vec{A}\to o}$.

Before we define the semantics, we observe several properties of the domains and the operations we have introduced. First, the generalization of $(\cdot)\lfloor_r$ to higher orders preserves the properties of Lemma 4.

▶ **Lemma 7.** *For every type A, both $\mathcal{D}_A$ and $\mathcal{R}_A$ are finite complete lattices. When A is $A_1 \to \cdots \to A_l \to o$, $g \in \mathcal{R}_A$, $\vec{h} \in \mathcal{R}_{A_1} \times \cdots \times \mathcal{R}_{A_l}$ and $r, r_1, r_2 \in [m]$ then:*

- $(g\lfloor_{r_1})\lfloor_{r_2} = g\lfloor_{\max(r_1,r_2)}$;
- $(q, rk(q)) \in g\lfloor_r(\vec{h})$ *iff* $(q, \max(rk(q), r)) \in g(\vec{h})$.

For every g_1, g_2 in $\mathcal{R}_A$: $(g_1 \vee g_2)\lfloor_r = g_1\lfloor_r \vee g_2\lfloor_r$ and $(g_1 \wedge g_2)\lfloor_r = g_1\lfloor_r \wedge g_2\lfloor_r$.

We now extend to higher-orders the operations $(\cdot)^\partial$ and $(\cdot)\cdot r$ we have introduced in Section 2.4. These extensions use the same covariant pattern as the extensions of $(\cdot)\Downarrow_q$ and $(\cdot)\lfloor_r$; we first define the operations for objects of type o and then extend them to all higher types. For $g_0 \in \mathcal{R}_o$, $f_0 \in \mathcal{D}_o$, $g_1 \in \mathcal{R}_{A\to B}$, $f_1 \in \mathcal{D}_{A\to B}$ we have:

$$g_0^\partial = \{q : (q, rk(q)) \in g_0\} \qquad g_1^\partial(h) = (g_1(h))^\partial$$
$$f_0 \cdot r = \{(q, r) : q \in f_0, rk(q) \le r\} \qquad (f_1 \cdot r)(h) = (f_1(h)) \cdot r$$

Thus g^∂ converts an element of $\mathcal{R}_A$ to an element of $\mathcal{D}_A$, and $f \cdot r$ does the opposite.

▶ **Lemma 8.** *For every type A, every $f \in \mathcal{D}_A$, $g \in \mathcal{R}_A$, and $r \in [m]$, we have: $f \cdot r \in \mathcal{R}_A$, $g\lfloor_r \in \mathcal{R}_A$, and $g^\partial \in \mathcal{D}_A$.*

The *semantics of a term* M of some type A, under a given valuation v is denoted $[\![M, v]\!]$. It is an element of $\mathcal{D}_A$ provided v is defined for all free variables of M. As in Section 2.4, a valuation is a function assigning to a variable of type B an element of $\mathcal{R}_B$. The semantic clauses are those from Figure 4, so they are the same as for the order 1 case of Section 2.4. It remains to define the fixpoint:

$$[\![Y^A, v]\!]h = \mathrm{fix}(h, 0) = \nu f_0.h^\partial\Big(f_0 \cdot 0 \vee \bigvee_{i=1}^{m} \mathrm{fix}(h, i) \cdot i\Big) \ .$$

where for $l = 0, \ldots, m$ we define

$$\mathrm{fix}(h, l) = \sigma f_l. \ldots \mu f_1. \nu f_0.\ (h\restriction_l)^\partial\big(\bigvee_{i=0}^{l} f_i \cdot i \vee \bigvee_{i=l+1}^{m} \mathrm{fix}(h, i) \cdot i\big)\ .$$

We use σ to stand for μ or ν depending on whether l is odd or even, respectively.

The structure of this formula may be better visible if we look at $\mathrm{fix}(h, m)$, and assume that m is odd:

$$\mu f_m. \nu f_{m-1} \ldots \mu f_1. \nu f_0.\ (h\restriction_m)^\partial\big(\bigvee_{i=0}^{m} f_i \cdot i\big)\ .$$

So we see a rather expected alternation of least and greatest fixpoints, and inside the big brackets we see an operation of composing f_i's to one residual. This operation is of the same shape as in the clause for application. Observe that the expression $(\bigvee_{i=0}^{m} f_i \cdot i)$ considered as a function of $f_0, \ldots, f_m$ is a monotone function from $\mathcal{D}_A^{m+1}$ to $\mathcal{D}_A$. This remark together with Lemma 8 and the fact that $\mathcal{D}_A$ is a complete lattice explains why $\mathrm{fix}(h, l)$ is well-defined, for every l.

We state a couple of lemmas implying that what we have defined is indeed a model.

▶ **Lemma 9.** *For every type A, if f is in $\mathcal{R}_{A\to A}$ then for every $k, l \in [m]$:* (i) $\mathrm{fix}(f, l)$ *is in $\mathcal{D}_A$; and* (ii) $\mathrm{fix}(f\restriction_k, l) = \mathrm{fix}(f, \max(k, l))$.

The next lemma implies that for every term M of type A, the value $[\![M, \upsilon]\!]$ assigned by the semantics is indeed in $\mathcal{D}_A$.

Notation: We write $(\cdot)\restriction_q$ for $(\cdot)\restriction_{rk(q)}$.

▶ **Lemma 10.** *For every term M, every υ, and $\vec{f}$, of appropriate types:*
1. *If $\upsilon \le \upsilon'$ and $\vec{f} \le \vec{g}$ then $[\![M, \upsilon]\!]\vec{f} \le [\![M, \upsilon']\!]\vec{g}$.*
2. *For every $q \in Q$: $q \in [\![M, \upsilon]\!]\vec{f}$ iff $q \in [\![M, \upsilon]\!]\vec{f}\restriction_q$ iff $q \in [\![M, \upsilon\restriction_q]\!]\vec{f}\restriction_q$.*
3. *$[\![M, \upsilon]\!]$ and $\langle\!\langle M, \upsilon\rangle\!\rangle$ satisfy the* (`strat`) *property.*
4. *$\langle\!\langle M, \upsilon\restriction_q\rangle\!\rangle = \langle\!\langle M, \upsilon\rangle\!\rangle\restriction_q$.*

The above lemmas allow us to show that the interpretation of terms is invariant under $=_{\beta\delta\eta}$, or, put differently, that we have constructed a model of λY-calculus.

▶ **Proposition 11.** *For every M, N and υ, if $M =_{\beta\delta\eta} N$, then $[\![M, \upsilon]\!] = [\![N, \upsilon]\!]$ and $\langle\!\langle M, \upsilon\rangle\!\rangle = \langle\!\langle N, \upsilon\rangle\!\rangle$.*

It now remains to explain how this model is related to the acceptance of the Böhm trees of λY-terms by $\mathcal{A}$. This explanation is given by the following theorem which is the main result of the paper. Recall that we denote the empty valuation by $\emptyset$.

▶ **Theorem 12** (Correctness)**.** *For a given parity automaton $\mathcal{A}$, the semantics defined above is such that for every closed term M of type o and every state q of $\mathcal{A}$:*

$q \in [\![M, \emptyset]\!]$ *iff $\mathcal{A}$ accepts $BT(M)$ from state q.*

▶ **Example 13.** Continuing the example from page 236 we will calculate the value of the term $M^{o\to o} = Y(\lambda f x.a\, x\, (f(b\, x)))$. This term is a simplified version of the `map` function from the

Introduction, in the sense that it has a Böhm tree of a similar shape. In order to show that every path of $BT(Mc)$ contains only finitely many b's we show $q_0 \in [\![Mc, \emptyset]\!]$. In the first part of the example we have established $[\![a\, x\, (f(b\, x)), \upsilon]\!] = \{q_0\}$ where υ that maps x to $\{(q_1, 1)\}$ and f to the step function $\{(q_1, 1)\} \mapsto (q_0, 0)$. This implies that $[\![\lambda f x.a\, x(f(b\, x))]\!] \geq g$ where $g = (\{(q_1, 1)\} \mapsto \{(q_0, 0)\}) \mapsto \{(q_1, 1)\} \mapsto q_0$. We now compute $\mathrm{fix}(g, 0)$. We observe that $g(\top \cdot 0 \vee \bot \cdot 1) = \{(q_1, 1)\} \mapsto q_0$ and, $g(h \cdot 0 \vee \bot \cdot 1) = h$, for $h = \{(q_1, 1)\} \mapsto q_0$. Therefore $\nu g_0.g(g_0 \cdot 0 \vee \bot \cdot 1) = h$. Now, $g(h \cdot 0 \vee h \cdot 1) = h$ which implies that $\mu g_1.\nu g_0.g(g_0 \cdot 0 \vee g_1 \cdot 1) = h$. With this we have showed $[\![M, \emptyset]\!] \geq h$ which finally gives us $q_0 \in [\![Mc]\!]$.

4 Applications

The model construction we have presented allows us to derive a number of results on verification of higher-order schemes and the λY-calculus. Since the constructed model is finite, it implies the decidability of the model-checking problem [19]. More importantly, it implies the transfer theorem [21]. Actually this theorem is proved in op. cit. also for infinite terms. This cannot be done solely with the techniques in the present paper. The transfer theorem gives an effective reduction of the MSOL theory $BT(M)$ to the MSOL theory of the tree representation of M. The strength of the theorem lies in the fact that the reduction is uniform for all terms over a fixed set of variables and types.

A term can be represented as a tree with back edges: the nodes of the tree are labelled with the application symbol, the lambda abstraction, a variable, or a constant. The back edges go from occurrences of variables to their binding lambdas. This representation makes it rather clear what it means for a term to be a model of an MSOL formula [21]. We will use $\mathit{Terms}(\Sigma, \mathcal{T}, X)$ for the set of terms over a signature Σ, such that all their (free or bound) variables are from $\mathcal{X}$, and all their subterms have types in $\mathcal{T}$.

▶ **Theorem 14** ([21]). *Let Σ be a finite tree signature, $\mathcal{X}$ a finite set of typed variables, and $\mathcal{T}$ a finite set of types. For every MSOL formula φ one can effectively construct an MSOL formula $\widehat{\varphi}$ such that for every λY-term $M \in \mathit{Terms}(\Sigma, \mathcal{T}, \mathcal{X})$ of type o:*

$$BT(M) \vDash \varphi \qquad \textit{iff} \qquad M \vDash \widehat{\varphi}.$$

Proof. Let $\mathcal{A}$ be the automaton equivalent to φ. Consider the model $\mathcal{D}^{\mathcal{A}}$ given by Theorem 12. The model $\mathcal{D}^{\mathcal{A}}$ is finite in every type. So the set of possible semantical values of terms from $\mathit{Term}(\Sigma, \mathcal{T}, \mathcal{X})$ is finite.

There is a correspondence between subterms of the term and the nodes of the tree representation of the term. So the labelling assigning to a node of the tree representation the meaning of the subterm it represents is a colouring of the tree with colours from a finite set. Let us call it the *semantic colouring*. The next observation is that if we are given any colouring of the tree representation of a term with elements of the model then we can check if it is the semantic colouring by verifying some local constraints implied by the definition of the model. For example, the local constraints say that the meaning assigned to the node labelled by the application symbol is indeed the result of the application of the meaning assigned to the first child applied to the meaning assigned to the second child. This can be checked by a looking in a finite table. Now the desired MSOL formula can guess such a colouring of the tree representation of a term, verify that it satisfies the local constraints, and that the initial state of the automaton $\mathcal{A}$ belongs to the colour of the root node. ◀

This theorem implies the global model checking property [3]. In particular, a model clearly explains how to solve the synthesis problem from higher-order modules [21]. The

synthesized program is composed from modules using application. Since the set of modules is fixed and finite, we can evaluate the meaning of such a composition using a finite automaton. Thus the synthesis problem is reduced to the emptiness problem for finite automata on finite trees.

As described in [22], a model can be used to design program transformations. A general principle of such a transformation is that during evaluation the program "knows" what is its meaning in the model. Such a program, or in our case a term of λY-calculus, is called *reflective* [2]. This intuitive statement requires some explanation. What we mean is that when evaluating a term M we reach a head normal form, say bN_1N_2. Then b is a non-interpreted symbol that is output as the root of the tree $BT(M)$, and the evaluation process splits to evaluation of N_1 and N_2. While at the beginning we can simply calculate the semantics $[\![M]\!]$ in the model, it is the reflective program itself that needs to calculate $[\![N_1]\!]$ and $[\![N_2]\!]$. Interestingly, this general method of translating a term into a reflective term follows a simple inductive pattern. We refer to [22] for more details.

5 Conclusions

We have extended Scott models with ranks, and have shown that this extension recognizes all MSOL properties of λY-terms. The meaning of the fixpoint operator is an alternation of the least and the greatest fixpoints reminiscent to the fixpoint characterization of winning positions in a parity game. This is somehow expected since acceptance for parity automata is expressed in terms of existence of a strategy in a parity game.

The model construction reduces the higher-order verification problem to the evaluation problem. Surprisingly, even the problem of evaluating terms without fixpoints in a Scott model is not that well studied (cf. [26]). We believe that the evaluation problem can be an unifying algorithmic problem for many kinds of program analyses whose theoretical complexity is "sufficiently high" to justify a semantic approach. Verification of MSOL properties considered in this paper is one such case. The model we construct is essentially of the same size as the Scott model so the evaluation approach should be essentially as efficient as approaches based on intersection types refining simple types. Indeed, every step function in the model can be represented by such a type.

We hope that our result is a step towards understanding infinitary properties in the usual frameworks of semantics, and with this to extend semantic methods to reactive programs and their behaviors. We have tried here to make the presentation as concrete as possible. It is evident though that a more abstract description bringing out the structure of the model should be pursued. A more ambitious goal is to find an abstract description of models recognizing MSOL properties. Let us mention that the expressive power of Scott models with arbitrary (be it as combinations of least and greatest fixpoints, or other kinds of fixpoints) interpretations of fixpoints is unknown. In particular, we may wonder whether they capture properties beyond those expressible with parity automata.

References

1 K. Aehlig. A finite semantics of simply-typed lambda terms for infinite runs of automata. *Logical Methods in Computer Science*, 3(1):1–23, 2007.

2 C. Broadbent, A. Carayol, L. Ong, and O. Serre. Recursion schemes and logical reflection. In *LICS*, pages 120–129, 2010.

3 C. Broadbent and C.-H. L. Ong. On global model checking trees generated by higher-order recursion schemes. In *FOSSACS*, volume 5504 of *LNCS*, pages 107–121, 2009.

4 C. H. Broadbent, A. Carayol, M. Hague, and O. Serre. C-shore: a collapsible approach to higher-order verification. In *ICFP*, pages 13–24. ACM, 2013.

5 Werner Damm. The IO- and OI-hierarchies. *Theoretical Computer Science*, 20:95–207, 1982.

6 J. Engelfriet and E. M. Schmidt. IO and OI. I. *Journal of computer and system sciences*, 15:328–353, 1977.

7 J. Engelfriet and E. M. Schnidt. IO and OI. II. *Journal of computer and system sciences*, 16:67–99, 1978.

8 Charles Grellois and Paul-André Melliès. An infinitary model of linear logic. In *FOSSACS 15*, volume 9034 of *LNCS*, pages 41–55, 2015.

9 A. Haddad. Model checking and functional program transformations. In *FSTTCS*, volume 24 of *LIPIcs*, pages 115–126, 2013.

10 M. Hofmann and W. Chen. Abstract interpretation from büchi automata. In *LICS-CSL*, pages 51:1–51:10, 2014.

11 Yu I Ianov. The logical schemes of algorithms. *Problems of cybernetics*, 1:82–140, 1960.

12 A Jeffrey. Functional reactive types. In *CSL-LICS*, pages 54:1–54:10, 2014.

13 Teodor Knapik, Damian Niwinski, and Pawel Urzyczyn. Higher-order pushdown trees are easy. In *FoSSaCS*, volume 2303, pages 205–222, 2002.

14 N. Kobayashi. Model checking higher-order programs. *J. ACM*, 60(3):20–89, 2013.

15 N. Kobayashi and L. Ong. A type system equivalent to modal mu-calculus model checking of recursion schemes. In *LICS*, pages 179–188, 2009.

16 P.A. Mellies. Linear logic and higher-order model checking, June 2014. http://www.pps.univ-paris-diderot.fr/~mellies/slides/workshop-IHP-model-checking.pdf.

17 P.A. Mellies. private communication, June 2014.

18 M. Naik and J. Palsberg. A type system equivalent to a model checker. *ACM Trans. Program. Lang. Syst.*, 30(5), 2008.

19 C.-H. L. Ong. On model-checking trees generated by higher-order recursion schemes. In *LICS*, pages 81–90, 2006.

20 S. J. Ramsay, R. P. Neatherway, and C.-H. L. Ong. A type-directed abstraction refinement approach to higher-order model checking. In *POPL*, pages 61–72. ACM, 2014.

21 S. Salvati and I. Walukiewicz. Evaluation is MSOL-compatible. In *FSTTCS*, volume 24 of *LIPIcs*, pages 103–114, 2013.

22 S. Salvati and I. Walukiewicz. Using models to model-check recursive schemes. In *TLCA*, volume 7941 of *LNCS*, pages 189–204, 2013.

23 Sylvain Salvati. Recognizability in the Simply Typed Lambda-Calculus. In *WOLIC*, volume 5514 of *LNCS*, pages 48–60, 2009.

24 Sylvain Salvati and Igor Walukiewicz. A model for behavioural properties of higher-order programs, April 2015. https://hal.archives-ouvertes.fr/hal-01145494.

25 Sylvain Salvati and Igor Walukiewicz. Typing weak MSOL properties. In *FOSSACS 15*, volume 9034, pages 343–357, 2015.

26 K. Terui. Semantic evaluation, intersection types and complexity of simply typed lambda calculus. In *RTA*, volume 15 of *LIPIcs*, pages 323–338. Schloss Dagstuhl, 2012.

27 T. Tsukada and C.-H. L. Ong. Compositional higher-order model checking via ω-regular games over Böhm trees. In *LICS-CSL*, pages 78:1–78:10, 2014.

Reachability Analysis of First-order Definable Pushdown Systems

Lorenzo Clemente* **and Sławomir Lasota†**

University of Warsaw, Poland

Abstract

We study pushdown systems where control states, stack alphabet, and transition relation, instead of being finite, are first-order definable in a fixed countably-infinite structure. We show that the reachability analysis can be addressed with the well-known saturation technique for the wide class of *oligomorphic structures*. Moreover, for the more restrictive *homogeneous structures*, we are able to give concrete complexity upper bounds. We show ample applicability of our technique by presenting several concrete examples of homogeneous structures, subsuming, with optimal complexity, known results from the literature. We show that infinitely many such examples of homogeneous structures can be obtained with the classical *wreath product* construction.

1998 ACM Subject Classification F.1.1 [Computation by Abstract Devices] Models of Computation, F.2.2 [Nonnumerical Algorithms and Problems] Computations on discrete structures, F.3.1 [Specifying and Verifying and Reasoning about Programs] Mechanical verification, F.4.1 [Mathematical Logic] Logic and constraint programming

Keywords and phrases automata theory, pushdown systems, sets with atoms, saturation technique

Digital Object Identifier 10.4230/LIPIcs.CSL.2015.244

1 Introduction

Context. Pushdown automata (PDS) are a well-known model of recursive programs, with applications in areas as diverse as language processing, data-flow analysis, security, computational biology, and program verification. Many interesting analyses reduce to checking reachability in the infinite configuration graph generated by a PDS, which can be done in PTIME with the popular *saturation algorithm* [7, 18] (cf. also the recent survey [11]). Saturation shows a slightly more general property of PDS graphs, which is sometimes called *effective preservation of regularity*: For a regular set of target configurations of a given PDS, the set of all configurations which can reach the target in a finite number of steps is effectively regular too. The preservation is *effective* in the sense that there exists a procedure which produces, from an NFA recognizing the target set, an NFA recognizing the predecessors. This is a central theoretical result in the analysis of PDS, with immediate practical applications as demonstrated by the prominent tool MOPED [17]. Therefore, it is of interest to extend this conceptually simple and yet powerful method to more general settings.

Several generalizations of the pushdown structure yielding PDS-like models admitting effective preservation of regularity are known, e.g., tree-pushdown systems [20], ordered multi-pushdown systems [9, 4], annotated higher-order pushdown systems [25, 10], and

* The first author acknowledges a partial support of the Polish National Science Centre grant 2013/09/B/ST6/01575.

† The last author acknowledges a partial support of the Polish National Science Centre grant 2012/07/B/ST6/01497.

24th EACSL Annual Conference on Computer Science Logic (CSL 2015).
Editor: Stephan Kreutzer; pp. 244–259

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

strongly normed multi-pushdown systems [14]. In this paper, instead of generalizing the pushdown structure itself, we generalize the *contents* of the pushdown, by allowing the pushdown symbols to be drawn from an infinite set. Our model is parametric in the choice of a countably-infinite logical structure $\mathbb{A}$, called *atoms*. We introduce and study *first-order definable pushdown systems* (FO-definable PDS) over $\mathbb{A}$, which are like usual PDS, except that control locations, stack alphabet, and transition relation are FO-definable sets over $\mathbb{A}$, instead of ordinary finite sets. Thus, we do not invent a new model, but we reinterpret the classical model in a new setting. This covers ordinary PDS as a special case, and allows the study of non-trivial yet decidable classes of PDS over infinite alphabets. For instance, by taking $\mathbb{A}$ to be *equality atoms* $(\mathbb{D}, =)$, i.e., a countably-infinite set $\mathbb{D}$ where only equality testing is allowed, we obtain (and slightly generalize) pushdown register automata [12, 6, 26].

Contributions and organization. The technical results of this paper and its structure are as follows. In Sec. 2, we recall the setting of FO-definable sets, FO-definable relations, and FO-definable NFA. In Sec. 3, we introduce FO-definable PDS. This is done by reinterpreting the classical model in the FO-definable framework. Our approach has the advantage that we do not need to define a new model. Instead, we *reinterpret* the classical model in a generic logical framework. In Sec. 4, we consider *oligomorphic atoms* [1] with a decidable first-order theory, and we show effective preservation of regularity for the backward reachability relation of configuration graphs of FO-definable PDS. This is obtained via a symbolic implementation of the classical saturation method, which comes along with a simple proof of correctness. In Sec. 5, we provide an upper complexity bound in the special case of *homogeneous* atoms, and in particular an `ExpTime` bound in the case of *tractable* homogenous atoms, matching the known `ExpTime`-hardness for equality atoms from [26]. In Sec. 6, we provide many interesting examples of tractable homogeneous atoms for which we can apply our results, including equality atoms [26] (as remarked above), but also: *total order atoms* $(\mathbb{Q}, \leq)$, which can be used for modeling densely-ordered data values; *equivalence atoms* $(\mathbb{D}, R)$, where R is an equivalence relation of infinite index s.t. each equivalence class is infinite, which can be used to model nested data values; *universal tree atoms*, which can be used to model dynamic topologies of concurrent programs with process creation and termination; as well as other structures, such as *universal partial order atoms*, *universal tournament atoms*, and *universal graph atoms* [24]. In the same section, we also show that the classic *wreath product* construction can be used to generate infinitely many new tractable examples from previous ones. Our logical approach has the advantage to highlight the general principle behind decidability, and we can thus prove correctness once and for all for *all* structures satisfying the mild assumptions above. As a byproduct, we also obtain tight complexity results for PDS over natural classes of infinite alphabets. Infinitely many such natural structures can be found by using the wreath product construction. In Sec. 7, we conclude with some directions for future work.

2 Preliminaries

Sets with atoms. Let $\mathbb{A}$ be a countably-infinite logical structure with finite vocabulary. An element of the structure we call *atom*, and the whole structure we call *atoms*. Examples of atoms are equality atoms $(\mathbb{D}, =)$, i.e., an arbitrary countable infinite set $\mathbb{D}$ with equality, and total order atoms $(\mathbb{Q}, \leq)$, i.e., the rationals with the dense order. More examples of atoms

[1] A structure $\mathbb{A}$ is *oligomorphic* if for every n, the product $\mathbb{A}^n$ is orbit-finite.

will be discussed in Sec. 6. In the study of atoms, the group $\mathrm{Aut}(\mathbb{A})$ of automorphisms[2] of $\mathbb{A}$ plays a central role. For instance, automorphisms of equality atoms are all permutations of $\mathbb{D}$, and automorphisms of total order atoms are monotonic permutations of $\mathbb{Q}$. By using atoms, we can build sets containing either previously built sets, or atoms themselves. For example, we build tuples $\mathbb{A}^n$ of fixed length, or disjoint unions thereof. On such sets, we will consider the natural action of $\mathrm{Aut}(\mathbb{A})$, which renames atoms while keeping intact the remaining structure. For instance, on tuples of atoms the natural action is the point-wise renaming: for $\pi \in \mathrm{Aut}(\mathbb{A})$ and $a_1, \ldots, a_n \in \mathbb{A}$, $\pi(a_1, \ldots, a_n) = (\pi(a_1), \ldots, \pi(a_n))$. Similarly, on disjoint unions the action is component-wise. The action induces the notion of *orbit*, which is the set of elements that can be reached via renaming, i.e., $\mathrm{orbit}(e) = \{\pi(e) \mid \pi \in \mathrm{Aut}(\mathbb{A})\}$. The sets in the sequel will always be *equivariant*, i.e., invariant under action of automorphisms[3]. Every orbit is equivariant by definition, and every equivariant set is a disjoint union of orbits. For instance, in total order atoms $(\mathbb{Q}, \leq)$, the set $\mathbb{Q}^2$ is the disjoint union of 3 orbits, $\{(q, q') \mid q < q'\}$, $\{(q, q') \mid q = q'\}$, and $\{(q, q') \mid q > q'\}$; and $\mathbb{Q}^2 \uplus \mathbb{Q}^3$ is the disjoint union of 16 orbits. A central notion is that of *orbit-finite* sets, which are *finite* unions of orbits (as opposed to arbitrary unions). Intuitively, an orbit-finite set has only finitely many elements up to renaming by atom automorphisms. Orbit-finiteness generalizes finiteness, and a substantial portion of results from automata theory carry over to the more general orbit-finite setting [5]. This paper can be seen as such a case study for the specific case of pushdown automata. For the sake of concreteness, we restrict in the rest of the paper to *FO-definable sets*, to be defined now; we only note that the results of this paper can be straightforwardly generalized to all orbit-finite sets with atoms.

FO-definable sets. Fix a structure $\mathbb{A}$ over a finite vocabulary. We describe infinite sets symbolically using first-order logic over the vocabulary of $\mathbb{A}$, which we assume to always include the equality relation $=$. A first-order formula $\varphi(\vec{x})$ (where we explicit list all free variables according to an implicit order) with $n \geq 1$ free variables *defines* the subset $[\varphi] \subseteq \mathbb{A}^n$ of tuples that satisfy φ, i.e., $[\varphi] = \{\vec{a} \in \mathbb{A}^n \mid (\vec{x} \mapsto \vec{a}) \vDash \varphi\}$. This set is always equivariant, since a formula can only compare atoms by using symbols from the signature, and automorphisms by definition respect this signature. The *dimension* of $[\varphi]$ is the number $n \geq 1$ of free variables of φ, denoted by $\dim \varphi$. We also allow the tautologically true formula $\varphi \equiv (\forall x \cdot x = x)$; by convention, we take $\dim \varphi = 0$ and $[\varphi]$ is a singleton (for a fixed atom in $\mathbb{A}$). A *FO-definable set X over* $\mathbb{A}$ is a finite indexed union of such sets, i.e.,

$$X = \bigcup_{l \in L} \{l\} \times [\varphi_l], \qquad \text{where } L \text{ is a finite index set.}$$

When we want to omit the formal indexing, we just write X as the finite disjoint union $\biguplus_{l \in L} [\varphi_l]$. Since FO-definable sets are unions of equivariant sets, they are equivariant too. When $\dim \varphi_l = 0$ for every $l \in L$, then X is finite and has the same number of elements as L. Thus, FO-definable sets generalize finite sets.

We use FO-definable sets for control locations and alphabets of automata. In the former case, an index $l \in L$ may be understood as a control location, and a tuple $\vec{a} \in \mathbb{A}^n$ as a valuation of n registers. Under this intuition, φ_l is an invariant that constrains register

[2] An *automorphism* is a bijection of atoms that preserves all relations from the vocabulary.

[3] More generally, one can consider *finitely supported* sets. A set is supported by $S \subseteq_{\mathrm{fin}} \mathbb{A}$ if it is invariant under automorphisms that preserve elements of S. The results of this paper can be straightforwardly generalized to finitely supported sets.

valuations in a control location l. We do not assume that all component sets $[\varphi_l]$ have the same dimension, i.e., the number of registers may vary from one control location to another.

FO-definable relations. Along the same lines, we define FO-definable binary relations. Consider two FO-definable sets $X = \biguplus_{l\in L}[\varphi_l]$ and $Y = \biguplus_{k\in K}[\psi_k]$. An FO-definable relation $R \subseteq X \times Y$ is an FO-definable set $R = \biguplus_{l\in L,k\in K}[\xi_{lk}]$ where the indexing set is the Cartesian product $L \times K$, and every component set $[\xi_{lk}]$ satisfies $[\xi_{lk}] \subseteq [\varphi_l] \times [\psi_k]$. In particular, $\dim \xi_{lk} = \dim \varphi_l + \dim \psi_k$. Relations of greater arities can be obtained by iterating the construction above. We use FO-definable relations to define transition relations of automata. The formula ξ_{lk} may be understood as a constraint on a transition from control location l to control location k, prescribing how a valuation of registers in l before the transition relates to a valuation of registers in k after the transition.

FO-definable NFA. As an example application of FO-definable sets and relations, we define FO-definable NFA. This model will be used later to recognize regular set of configurations of FO-definable PDS, also defined later. A classical NFA is a tuple $\mathcal{A} = (\Gamma, Q, F, \delta)$, where Γ is a finite input alphabet, Q is a finite set of states, of which those in $F \subseteq Q$ are the final ones, and $\delta \subseteq Q \times \Gamma \times Q$ is the transition relation. Once an initial state is chosen, the definitions of run, accepting run, and language $\mathcal{L}(\mathcal{A})$ recognized by $\mathcal{A}$ are standard. By simply replacing "finite" with "FO-definable" in the definition above, we obtain FO-definable NFA. To fix notation, an FO-definable NFA will be written as a tuple $\mathcal{A} = (\Gamma = \biguplus_{k\in K}[\varphi_k],\ Q = \biguplus_{l\in L}[\psi_l],\ F = \biguplus_{l\in L}[\psi^F_l],\ \delta = \biguplus_{l,l'\in L,k\in L}[\delta_{lkl'}])$, where w.l.o.g. we assume that Q and F have the same index set L. Notice that δ is an FO-definable set, while $\delta_{lkl'}$ is a first-order formula.

▶ **Example 1.** Let $\mathbb{A}$ be the total order atoms $(\mathbb{Q}, \leq)$, and let the alphabet be $\Gamma = \{k\} \times \mathbb{Q}$. Consider the language $M = \{(k, a_1) \cdots (k, a_n) \in \Gamma^* \mid a_1 \geq a_2 \leq a_3 \geq \cdots \leq a_{2n+1}\}$ of non-empty finite words of odd length of alternating growth. This language can be recognized from state ℓ_I by the NFA

$$\mathcal{A} = (\Gamma,\ Q = \{\ell_I\} \cup \{\ell_0\} \times \mathbb{Q} \cup \{\ell_1\} \times \mathbb{Q}, F = \{\ell_0\} \times \mathbb{Q}, \delta = \biguplus_{l,l'\in\{\ell_I,\ell_0,\ell_1\}} [\delta_{lkl'}]).$$

The initial location ℓ_I does not contain any register, while control locations ℓ_0, ℓ_1 both contain one register, which is used to guess the next input symbol and to ensure the right ordering. Formally, $\delta_{\ell_I k \ell_0}(, y, x') \equiv x' \leq y$ (we use the notation $\delta_{\ell_I k \ell_0}(, y, x')$ to emphasize that ℓ_I does not have any register), $\delta_{\ell_0 k \ell_1}(x, y, x') \equiv (x = y \wedge x' \geq y)$, $\delta_{\ell_1 k \ell_0}(x, y, x') \equiv (x = y \wedge x' \leq y)$, and $[\delta_{lkl'}] = \emptyset$ for the other cases.

3 First-order definable pushdown systems

In this section we define FO-definable PDS and their reachability problem. According to the classical definition, a pushdown system (PDS) $\mathcal{P} = \langle \Gamma, P, \rho \rangle$ consists of a finite stack alphabet Γ, a finite set of control states P, and a finite set of transition rules $\rho = \rho^{\mathsf{push}} \cup \rho^{\mathsf{pop}}$, which is partitioned into push rules $\rho^{\mathsf{push}} \subseteq P \times \Gamma \times P \times \Gamma \times \Gamma$ and pop rules $\rho^{\mathsf{pop}} \subseteq P \times \Gamma \times P$. In this paper, we reinterpret this definition in the setting of FO-definable sets, which yields a more general model. For an atom structure $\mathbb{A}$, *FO-definable PDS over* $\mathbb{A}$ are obtained by replacing "finite set" with "FO-definable set" in the classical definition. To fix notation, an

FO-definable PDS is a tuple

$$\mathcal{P} = \langle \Gamma = \biguplus_{k \in K} [\varphi_k],\ P = \biguplus_{\ell \in L} [\xi_\ell],\ \rho = \rho^{\mathsf{push}} \cup \rho^{\mathsf{pop}} \rangle,$$

where[4] $\rho^{\mathsf{push}} = \biguplus_{\ell,\ell' \in L, k, k', k'' \in K} [\rho^{\mathsf{push}}_{\ell k \ell' k' k''}]$ and $\rho^{\mathsf{pop}} = \biguplus_{\ell,\ell' \in L, k \in K} [\rho^{\mathsf{pop}}_{\ell k \ell'}]$. As in the classical case, an FO-definable PDS induces an infinite transition system $\langle \mathcal{C}, \longrightarrow \rangle$, where the set of configurations is $\mathcal{C} = P \times \Gamma^*$, and there is a transition $c \longrightarrow c'$ between two configurations $c = (q, aw)$ and $c' = (q', w')$ if, and only if, either there exists a push rule $(q, a, q', b, c) \in \rho^{\mathsf{push}}$ s.t. $w' = bcw$, or there exists a pop rule $(q, a, q') \in \rho^{\mathsf{pop}}$ s.t. $w = w'$. Let $\longrightarrow^*$ be the reflexive and transitive closure of $\longrightarrow$. For a set C of configurations, the *backward reachability set of* C, denoted $\mathrm{Reach}^{-1}_{\mathcal{P}}(C)$, is the set of configurations that can reach some configuration in C:

$$\mathrm{Reach}^{-1}_{\mathcal{P}}(C) = \{c \in \mathcal{C} \mid c \longrightarrow^* c' \text{ for some } c' \in C\} \ .$$

▶ **Example 2.** We define an FO-definable PDS $\mathcal{P}$ over total order atoms $(\mathbb{Q}, \leq)$ which constructs strictly monotonic stacks, the maximal element being on the top of the stack. Let $\mathcal{P} = \langle \Gamma = \{k\} \times \mathbb{Q}, P = \{\ell_I\}, \rho = \rho^{\mathsf{push}} \rangle$, where $\rho^{\mathsf{push}}_{\ell_I k \ell_I k k}(, y, , y', y'') \equiv (y < y' \wedge y'' = y)$.

This paper concentrates on the reachability analysis for FO-definable PDS. Given an FO-definable PDS $\mathcal{P} = \langle \Gamma, P, \rho \rangle$, two control locations $p, q \in P$, and a stack symbol $\bot \in \Gamma$, the *reachability problem* asks whether $(p, \bot) \in \mathrm{Reach}^{-1}_{\mathcal{P}}(\{q\} \times \Gamma^*)$. We start with stack $\bot$ and we ignore the stack at the end of the computation. More general analyses can be considered by imposing regular constraints on the initial and final stack contents. These easily reduce to reachability of a regular set of configurations, which is the problem considered in the next section.

4 Preservation of regularity I: Oligomorphic atoms

We solve the reachability problem as a corollary of a general effective preservation of regularity result for the backward reachability relation of FO-definable PDS. To this end, we use FO-definable NFA to describe regular sets of configurations. In the following, fix an FO-definable PDS $\mathcal{P} = \langle \Gamma, P, \rho \rangle$, and an FO-definable NFA $\mathcal{A} = \langle \Gamma, Q, F, \delta \rangle$ s.t. $P \subseteq Q$. The NFA $\mathcal{A}$ recognizes the following language $L_{\mathcal{P}}(\mathcal{A})$ of configurations of $\mathcal{P}$,

$$\mathcal{L}_{\mathcal{P}}(\mathcal{A}) = \{(p, w) \in P \times \Gamma^* \mid \mathcal{A} \text{ accepts } w \text{ from state } p\} \, .$$

Such sets of configurations of $\mathcal{P}$ we call *regular*. We assume w.l.o.g. that states of $\mathcal{A}$ that belong to P do not have incoming transitions, i.e. $\delta \subseteq Q \times \Gamma \times (Q \setminus P)$.

▶ **Example 3.** Recall the FO-definable PDS $\mathcal{P}$ from Example 2 building strictly monotonic stacks (maximal element on top). Let N be the following set of configurations

$$N = \{(\ell_I, (k, a_1) \cdots (k, a_{2n+1})) \in P \times \Gamma^* \mid a_1 \geq a_2 \leq a_3 \geq \cdots \leq a_{2n+1}\} \, .$$

This set is regular, and it is recognized by the NFA $\mathcal{A}$ from Example 1, i.e., $\mathcal{L}_{\mathcal{P}}(\mathcal{A}) = N$. The backward reachability set is

$$\mathrm{Reach}^{-1}_{\mathcal{P}}(N) = N \cup \{(\ell_I, (k, a_2) \cdots (k, a_{2n+1})) \in P \times \Gamma^* \mid a_2 \leq a_3 \geq \cdots \leq a_{2n+1}\} \, .$$

We will see below how to compute an FO-definable NFA recognizing $\mathrm{Reach}^{-1}_{\mathcal{P}}(N)$.

[4] We could have also considered push rules which do not read the top of the stack, i.e., of the form $\rho^{\mathsf{push}} = \biguplus_{\ell,\ell' \in L, k' \in K} [\rho^{\mathsf{push}}_{\ell \ell' k'}]$. However, these would introduce ϵ-transitions during our saturation procedure in Sec. 4, which we want to avoid for simplicity.

(0) $\delta' := \delta \cup \rho^{\text{pop}}$
(1) `repeat`
(2) $\quad \delta' := \delta' \cup \text{forced}(\delta')$
(3) `until` $\text{forced}(\delta') \subseteq \delta'$

Figure 1 Abstract saturation algorithm.

We solve the reachability problem for PDS over *oligomorphic* atoms.[5]. Oligomorphicity is an important notion in model theory [24]. Formally, a structure is oligomorphic if, and only if, for every $n \in \mathbb{N}$, the set $\mathbb{A}^n$ is orbit-finite. Not all structures are oligomorphic, as shown in the following example.

▶ Remark (Timed atoms). Timed atoms $(\mathbb{Q}, \leq, +1)$ is a well-known example of non-oligomorphic structure. They extend total order atoms $(\mathbb{Q}, \leq)$ with the successor relation $(+1) \subseteq \mathbb{Q} \times \mathbb{Q}$. Automorphisms of timed atoms are monotone bijections π of $\mathbb{Q}$ that preserve unit intervals, i.e., $\pi(x+1) = \pi(x) + 1$. To see why timed atoms are non oligomorphic, it suffices to see that already $\mathbb{Q}^2$ has infinitely-many orbits. Indeed, for each $z \in \mathbb{Z}$, $\mathbb{Q}^2$ has a disjoint orbit $\{(x, y) \in \mathbb{Q}^2 \mid x - y = z\}$. (Since automorphisms preserve unit intervals, they preserve all integer distances.) Working in non-oligomorphic structures like timed atoms requires the use of specialized techniques, and the generic algorithm presented in this section does not terminate. We have thoroughly studied the reachability problem for FO-definable pushdown systems and automata over timed atoms in [13].

Since oligomorphic atoms are very general, we can merely state decidability of the reachability problem, without any complexity bounds. The only additional assumption that we require is decidability of the *first-order satisfiability problem* in the structure $\mathbb{A}$, which asks, given a first-order formula $\varphi(x_1, \ldots, x_n)$, whether some valuation $\eta : \{x_1, \ldots, x_n\} \to \mathbb{A}$ of its free variables satisfies φ.

▶ **Theorem 4.** *Let $\mathbb{A}$ be an oligomorphic structure with a decidable first-order satisfiability problem. For FO-definable PDS $\mathcal{P}$ over $\mathbb{A}$ and an FO-definable NFA $\mathcal{A}$ over $\mathbb{A}$ recognizing a regular set of configurations $L_{\mathcal{P}}(\mathcal{A})$, one can effectively construct an FO-definable NFA $\mathcal{B}$ over $\mathbb{A}$ recognizing $L_{\mathcal{P}}(\mathcal{B}) = Reach_{\mathcal{P}}^{-1}(L_{\mathcal{P}}(\mathcal{A}))$.*

We prove Theorem 4 by using the classical *saturation technique* [7, 18]. We first describe a simple abstract algorithm manipulating infinite sets of transitions, and then we show how this can be implemented symbolically at the level of formulas. As in the classical case, the FO-definable NFA $\mathcal{B}$ which is computed by the algorithm is of the form $\langle \Gamma, Q, F, \delta' \rangle$ with $\delta \subseteq \delta'$, i.e., it is obtained by adding certain transitions to $\mathcal{A}$. For any relation $\alpha \subseteq Q \times \Gamma \times Q$, let $\text{forced}(\alpha) \subseteq Q \times \Gamma \times Q$ be the following set of triples:

$$\text{forced}(\alpha) = \left\{(q, a, q') \mid \exists (q, a, q'', b, c) \in \rho^{\mathsf{push}}, \exists (q'', b, q''') \in \alpha, \exists (q''', c, q') \in \alpha\right\}.$$

The abstract saturation algorithm is shown in Fig. 1.

[5] One could also consider PDS defined by general prefix rewriting, i.e., with transitions in $\rho \subseteq P \times \Gamma^* \times P \times \Gamma^*$. For oligomorphic atoms, our simplified push/pop model can simulate prefix rewriting while preserving reachability properties (but not configuration graph isomorphism, or even bisimilarity), like in the classical case.

INPUT: an FO-definable PDS $\mathcal{P} = \langle \Gamma = \biguplus_k [\varphi_k], P = \biguplus_\ell [\xi_\ell], \rho^{\mathsf{push}} \cup \rho^{\mathsf{pop}} \rangle$, with

$$\rho^{\mathsf{push}} = \biguplus_{\ell k \ell' k' k''} [\rho^{\mathsf{push}}_{\ell k \ell' k' k''}], \rho^{\mathsf{pop}} = \biguplus_{\ell k \ell'} [\rho^{\mathsf{pop}}_{\ell k \ell'}], \text{ and an FO-definable NFA}$$

$$\mathcal{A} = \langle \Gamma, Q = \biguplus_\ell [\psi_\ell], \delta = \biguplus_{\ell k \ell'} [\delta_{\ell k \ell'}] \rangle, \text{ with } [\xi_\ell] \subseteq [\psi_\ell], \text{ for every } \ell \in L.$$

(0) for every $\ell, k, \ell' :\ \delta'_{\ell k \ell'}(\vec{x}, \vec{y}, \vec{x}') \ :=\ \delta_{\ell k \ell'}(\vec{x}, \vec{y}, \vec{x}') \ \vee\ \rho^{\mathsf{pop}}_{\ell k \ell'}(\vec{x}, \vec{y}, \vec{x}')$

(1) `repeat`

(2) for every $\ell, k, \ell' :\ \delta'_{\ell k \ell'}(\vec{x}, \vec{y}, \vec{x}') \ :=\ \delta'_{\ell k \ell'}(\vec{x}, \vec{y}, \vec{x}') \ \vee\ \text{forced}(\delta')_{\ell k \ell'}(\vec{x}, \vec{y}, \vec{x}')$

(3) `until`$\big(\bigwedge_{\ell,k,\ell'} \forall \vec{x}, \vec{y}, \vec{x}' \cdot \text{forced}(\delta')_{\ell k \ell'}(\vec{x}, \vec{y}, \vec{x}') \implies \delta'_{\ell k \ell'}(\vec{x}, \vec{y}, \vec{x}') \big)$

Figure 2 Concrete saturation algorithm; ℓ, ℓ' range over L, and k ranges over K.

The algorithm is partially correct for every structure $\mathbb{A}$ (even though it might not terminate). This follows directly from the observation that the saturated NFA $\mathcal{B}$ has a transition $(q, a, q') \in \delta'$ between states $q, q' \in P$ of $\mathcal{P}$ if, and only if, $\mathcal{P}$ admits a run $(q, a) \longrightarrow^* (q', \varepsilon)$ (we use here the assumption that no transition of $\mathcal{A}$ ends in a state $q \in P$ of $\mathcal{P}$). However, on arbitrary structures saturation does not terminate, either because the inclusion checking on line (3) is not decidable, or because it never actually holds. The first issue is addressed by the requirement that $\mathbb{A}$ has a decidable first-order satisfiability problem, and the second one by the fact that $\mathbb{A}$ is an oligomorphic structure.

We implement the abstract algorithm from Fig. 1 symbolically, by manipulating formulas instead of actual transitions. We assume w.l.o.g. that the index set of P (the control locations of $\mathcal{P}$) is the same as the index set of Q (the states of $\mathcal{A}$). First, notice that the set $\text{forced}(\alpha)$ is FO-definable whenever α is so, since it can be expressed as follows:

$$\text{forced}(\alpha)_{\ell k \ell'}(\vec{x}, \vec{y}, \vec{x}') := \bigvee_{\ell'', \ell''' \in L, k', k'' \in K} \exists \vec{x}'', \vec{y}', \vec{y}'', \vec{x}''' \cdot \rho^{\mathsf{push}}_{\ell k \ell'' k' k''}(\vec{x}, \vec{y}, \vec{x}'', \vec{y}', \vec{y}'') \wedge \alpha_{\ell'' k' \ell'''}(\vec{x}'', \vec{y}', \vec{x}''') \ \wedge\ \alpha_{\ell''' k'' \ell'}(\vec{x}''', \vec{y}'', \vec{x}'),$$

where L is the index set of Q, and K is the index set of Γ. Steps (0) (initialization of δ') and (2) (update of δ') of the algorithm are implemented by disjunction of FO-definable sets, therefore at each stage of the algorithm δ' is an FO-definable set, and thus an equivariant set (i.e, a union of orbits). The test (3) is computable whenever first order satisfiability is so. We obtain the concrete algorithm in Fig. 2. Termination is guaranteed since $\mathbb{A}$ is oligomorphic, which implies orbit-finiteness of $Q \times \Gamma \times Q$. Indeed, δ' is always a union of orbits at every stage, and therefore at least one orbit is added to δ' at every iteration.

▶ **Example 5.** We apply the concrete saturation algorithm to the PDS $\mathcal{P}$ and NFA $\mathcal{A}$ from Example 3. Recall that $\mathcal{P} = \langle \Gamma = \{k\} \cup \mathbb{Q}, P = \{\ell_I\}, \rho^{\mathsf{push}} \rangle$, with $\rho^{\mathsf{push}}_{\ell_I k \ell_I k k}(, y, , y', y'') \equiv (y < y' \wedge y'' = y)$, and $\mathcal{A} = \langle \Gamma, Q = \{\ell_I\} \cup \{\ell_0, \ell_1\} \times \mathbb{Q}, F = \{\ell_0\} \times \mathbb{Q}, \delta \rangle$, with $\delta_{\ell_I k \ell_0}(, y, x') \equiv x' \leq y$, $\delta_{\ell_0 k \ell_1}(x, y, x') \equiv (x = y \wedge x' \geq y)$, $\delta_{\ell_1 k \ell_0}(x, y, x') \equiv (x = y \wedge x' \leq y)$ (omitting the trivial cases). For the first iteration, let $\delta^0 := \delta$. We compute $\text{forced}(\delta^0)$, for which the only nontrivial case is $\text{forced}(\delta^0)_{\ell_I k \ell_1}(, y, x') \equiv \exists y', y'', x''' \cdot \rho^{\mathsf{push}}_{\ell_I k \ell_I k k}(, y, , y', y'') \wedge \delta^0_{\ell_I k \ell_0}(, y', x''') \wedge \delta^0_{\ell_0 k \ell_1}(x''', y'', x')$,

which equals

$$\exists y', y'', x''' \cdot (y < y' \wedge y'' = y) \wedge (x''' \leq y') \wedge (x''' = y'' \wedge x' \geq y'').$$

By removing quantifiers (thanks to the density of $\mathbb{Q}$), the former is equivalent to $x' \geq y$. Therefore, δ^1 extends δ^0 with the new transition $\delta^1_{\ell_I k \ell_1}(,y,x') \equiv (x' \geq y)$. Since δ^1 is not equivalent to δ^0, we go to the next iteration. We compute forced(δ^1), for which the only new case is forced$(\delta^1)_{\ell_I k \ell_0}(,y,x') \equiv \exists y', y'', x''' \cdot \rho^{\mathsf{push}}_{\ell_I k \ell_I k k}(,y,,y',y'') \wedge \delta^1_{\ell_I k \ell_1}(,y',x''') \wedge \delta^1_{\ell_1 k \ell_0}(x''',y'',x')$, which equals

$$\exists y', y'', x''' \cdot (y < y' \wedge y'' = y) \wedge (x''' \geq y') \wedge (x''' = y'' \wedge x' \leq y'').$$

The latter is equivalent to $\exists y' \cdot y < y' \wedge y \geq y' \wedge x' \leq y$, which is clearly unsatisfiable. Therefore δ^2 is equivalent to δ^1, and the algorithms stops. It is immediate to check that $\mathcal{B} = \langle \Gamma, Q = \ell_I \cup \{\ell_0, \ell_1\} \times \mathbb{Q}, F = \{\ell_0\} \times \mathbb{Q}, \delta^1 \rangle$ recognizes precisely $\mathrm{Reach}^{-1}_{\mathcal{P}}(N)$, where $N = \mathcal{L}_{\mathcal{P}}(\mathcal{A})$.

5 Preservation of regularity II: Homogeneous atoms

Relational homogeneous structures are a well-behaved subclass of oligomorphic structures, for which we are able to give precise complexity upper bounds for our saturation construction. A relational structure $\mathbb{A}$ (i.e., with no function symbols in the vocabulary) is *homogeneous* if every isomorphism between two finite induced substructures[6] of $\mathbb{A}$ extends to an automorphism of the whole $\mathbb{A}$. This immediately implies that $\mathbb{A}$ is oligomorphic.

▶ **Proposition 1.** *Let $\mathbb{A}$ be a relational homogeneous structure. For $n \geq 1$, the number of orbits of $\mathbb{A}^n$ is bounded by $2^{poly(n)}$.*

Proof. A tuple of n elements $(a_1, \ldots, a_n) \in \mathbb{A}^n$ can be seen as an induced substructure of $\mathbb{A}$, where elements are additionally labelled with the positions $\{1 \ldots n\}$. Two such induced substructures $\bar{a}, \bar{b} \in \mathbb{A}^n$ are isomorphic exactly when the elements $\bar{a}$ and $\bar{b}$ satisfy the same relations in the vocabulary of $\mathbb{A}$. Therefore, there number of isomorphism classes is bounded by $2^{\mathrm{poly}(n)}$. Since $\mathbb{A}$ is homogeneous, every isomorphism between $\bar{a}$ and $\bar{b}$ extends to an automorphism of the whole $\mathbb{A}$, and thus $\bar{a}$ and $\bar{b}$ are in the same orbit. Consequently, the same bound applies to the number of orbits of $\mathbb{A}^n$. ◀

All structures listed in the introduction are homogeneous relational structures. However, not all oligomorphic relational structures are homogeneous as the example below shows.

▶ **Example 6** (Bit vector atoms)**.** Let a *bit vector* be any infinite sequence of zeros and ones with only finitely many ones. A bit vector can be represented by a finite sequence, by cutting off the infinite zero suffix. Consider the relational structure $\mathbb{V} = (V, 0, +)$, consisting of the set V of all bit vectors, together with a unary predicate $0(\_)$ that distinguishes the zero vector, and the ternary relation $\_ + \_ = \_$ that describes point-wise addition modulo 2. Automorphisms of $\mathbb{V}$ are precisely linear mappings, i.e., bijections f s.t. $f(0) = 0$ and $f(u + v) = f(u) + f(v)$. The orbit of a tuple $(v_1, \ldots, v_n) \in V^n$ is determined by its *addition type*, i.e., by the the set of all equalities of the form $v_{i_1} + \ldots + v_{i_m} = 0$ satisfied by $(v_1, \ldots, v_n)$. Indeed, for two tuples $(u_1, \ldots, u_n), (v_1, \ldots, v_n) \in V^n$ having the same addition type, consider the partial bijection f defined as $f(u_1) = v_1, \ldots, f(u_n) = v_n$. By using the Steinitz exchange

[6] An *induced substructure* is a structure obtained by restricting the universe to a subset of atoms.

lemma, the function f can be extended to a linear mapping on the whole V, and thus $(u_1, \ldots, u_n)$ and $(v_1, \ldots, v_n)$ are in the same orbit. Therefore, the number of orbits of V^n is finite. On the other hand, $\mathbb{V}$ is not homogeneous. For instance, the two induced substructures $X = \{1000, 0100, 0010, 0001\}$ and $Y = \{1000, 0100, 0010, 1110\}$ are isomorphic. Define, e.g., $f(0001) = 1110$, and $f(x) = x$ if $x \neq 0001$. The reason why f is an isomorphism is that f needs to respect $\_ + \_ = \_$ only inside its domain, and any combination of two vectors from X falls outside of X. However, the isomorphism f does not extend to an automorphism of $\mathbb{V}$, since vectors in Y are not independent[7].

It is worth mentioning that, while some atom structures are not homogenous, sometimes adding extra relational symbols (thus restricting the notion of isomorphic substructure) can make it homogeneous; cf. the example of universal tree order atoms from Sec. 6, where adding one extra relational symbol turns a non-homogeneous structure it into a homogeneous one.

Fix a homogeneous relational structure $\mathbb{A}$. We give a precise complexity upper-bound for the complexity of the concrete saturation procedure from Fig. 2 and, thus, for reachability. This depends on the complexity of the induced substructure problem for $\mathbb{A}$. The *(finite) induced substructure problem* for $\mathbb{A}$ asks whether a given finite structure A over the same vocabulary is an induced substructure of $\mathbb{A}$. This amounts to find an isomorphism mapping elements from A into atoms $\mathbb{A}$ s.t. all relations from the vocabulary are preserved. Assume that the induced substructure problem for $\mathbb{A}$ is decidable in time $T(k)$, where k is the size of the input. The complexity estimations below are always understood with respect to the sizes of the representing formulas. Let the *width* of a formula be the number of its variables. Let n be the width of an input automaton, defined as the greatest width of the formulas appearing in its definition, and let m be its *size*, defined as the sum of sizes of the defining formulas. By *T-relative pseudo-polynomial* time complexity we mean the time complexity

$$2^{\mathrm{poly}(n)} \cdot \mathrm{poly}(m) \cdot T(\mathrm{poly}(n)),$$

i.e., exponential in the width n but polynomial in the size m. Note that this is *relative* to the complexity T of the induced substructure problem.

▶ **Theorem 7.** *Let $\mathbb{A}$ be a homogeneous structure with induced substructure problem decidable in time $T(k)$. For FO-definable PDS $\mathcal{P}$ over $\mathbb{A}$ and an FO-definable NFA $\mathcal{A}$ recognizing a regular set of configurations $L_{\mathcal{P}}(\mathcal{A})$, one can construct in T-relative pseudo-polynomial time an FO-definable NFA $\mathcal{B}$ recognizing $L_{\mathcal{P}}(\mathcal{B}) = Reach_{\mathcal{P}}^{-1}(L_{\mathcal{P}}(\mathcal{A}))$.*

As a consequence, reachability in FO-definable PDS over $\mathbb{A}$ is decidable in T-relative pseudo-polynomial time.

Proof. Fix a homogeneous relational structure $\mathbb{A}$, and suppose that its induced substructure problem is decidable in time $T(k)$. We show that the concrete saturation algorithm from Fig. 2 terminates in T-relative pseudo-polynomial time. We use quantifier-free formulas over the vocabulary of $\mathbb{A}$ in *legal disjunctive normal form*, to be defined below. A *positive literal* is a predicate of the form $r(x_1, \ldots, x_k)$, where $x_1, \ldots, x_k$ are variables, and r is a relational

[7] The notion of homogeneity can be extended to structures with relations and functions, but one must consider *finitely-generated* induced substructures of $\mathbb{A}$ instead of finite ones. Note that $\mathbb{V}$ becomes homogeneous if $+$ is considered as a binary *function*, instead of a relation. The reason is that, in the presence of the functional symbol $+$, the homogeneity condition for $\mathbb{V}$ quantifies over finite induced substructures that are closed w.r.t. $+$, unlike the substructures in our example.

symbol in the vocabulary of $\mathbb{A}$. A *negative literal* is the negation $\neg r(x_1, \ldots, x_k)$ of a positive literal, and a *literal* is either a positive or a negative literal. We treat equality in the same way as other relations of $\mathbb{A}$, thus there are also equality and inequality literals. A *clause* is a conjunction of pairwise different literals. A clause φ is *complete* if, for every positive literal l over the variables of φ, either l or its negation appears in φ, but not both. A complete clause φ is *consistent* if

- the equality literals define an equivalence over the variables of φ, and
- the literals of φ are invariant under this equivalence relation, i.e., replacing variables appearing in a literal of φ with equivalent ones yields a literal that also appears in φ.

A consistent clause φ gives rise to a finite structure $\mathcal{A}_\varphi$ over the same vocabulary as $\mathbb{A}$, whose elements are equivalence classes of variables, and where a relation $r([x_1], \ldots, [x_k])$ holds if, and only if, $r(x_1, \ldots, x_k)$ appears in φ (the choice of representative variables is irrelevant since φ is consistent). Thus, valuations satisfying φ are in one-to-one correspondence with *embeddings* of $\mathcal{A}_\varphi$ into $\mathbb{A}$, by which we mean injective homomorphisms that both preserve and reflect relations. A consistent clause φ is *legal* if, and only if, the structure $\mathcal{A}_\varphi$ is isomorphic to an induced substructure of $\mathbb{A}$, i.e., if there exists an embedding of $\mathcal{A}_\varphi$ into $\mathbb{A}$, written $\mathcal{A}_\varphi \sqsubseteq \mathbb{A}$. Thus, a clause φ is legal if, and only if, it is satisfiable.

▶ **Proposition 2.** *Legality of a complete clause of size m is decidable in time $\mathrm{poly}(m) + T(m)$.*

We consider two clauses to be equal when they contain the same literals. A formula is in *legal disjunctive normal form (ldnf)* if it is a disjunction of pairwise different legal clauses over the same variables. We use the convention that the empty clause and the empty ldnf represent, respectively, true and false. For two formulas φ and ψ with the same free variables, we say that they are *equivalent*, written $\varphi \equiv \psi$, when $[\varphi] = [\psi]$, i.e., when they define the same set of tuples.

▶ **Proposition 3.** *A quantifier-free formula φ can be transformed into an equivalent formula ψ in ldnf in T-relative pseudo-polynomial time.*

Proof. Enumerate exhaustively all complete clauses over the variables of φ, and keep only those clauses $\{\psi_i\}_i$ which are legal (which is efficiently checkable by Proposition 2), and that satisfy φ (computable in time polynomial in the size of φ). Take $\psi = \bigvee_i \psi_i$. Clearly, $\psi \equiv \varphi$. The time complexity claim follows since the number of complete clauses is exponential in the number of variables, but independent from the size of φ. ◀

For homogeneous structures, the previous claim can be strengthened to first-order formulas. Essentially, this follows from the fact that, in a homogeneous structure, existential quantification can always be resolved positively.

▶ **Proposition 4.** *A first-order formula φ can be transformed to an equivalent formula ψ in ldnf in T-relative pseudo-polynomial time.*

Proof. As the first step, transform the input formula into prenex normal form. Then, transform the quantifier-free subformula into an equivalent ldnf, using Proposition 3. Finally, eliminate the quantifiers in sequence, starting from the innermost one, keeping the quantifier-free subformula in ldnf. Elimination of one existential quantifier is done as follows. First, distribute it over the disjunction of clauses,

$$\varphi \quad \equiv \quad \exists x \cdot \psi_1 \vee \ldots \vee \psi_n \quad \equiv \quad \exists x \cdot \psi_1 \ \vee \ \ldots \ \vee \ \exists x \cdot \psi_n$$

and then replace every disjunct $\exists x \cdot \psi_i$ with the clause ψ_i' obtained from ψ_i by removing those literals that contain x. We claim that, after elimination of duplicates,

$$\varphi \quad \equiv \quad \psi_1' \ \vee \ \ldots \ \vee \ \psi_{n'}' \ ,$$

where the right-hand side is in ldnf. To this end, we show that each ψ_i' is legal, and that $\exists x \cdot \psi_i \equiv \psi_i'$. Let $\mathcal{A}_{\psi_i}$ and $\mathcal{A}_{\psi_i'}$ be the two substructures of $\mathbb{A}$ defined by the two clauses. Clearly, $\mathcal{A}_{\psi_i'} \sqsubseteq \mathcal{A}_{\psi_i} \sqsubseteq \mathbb{A}$, which immediately implies legality of ψ_i' by transitivity. The left-to-right inclusion $[\exists x \cdot \psi_i] \subseteq [\psi_i']$ of the equivalence between $\exists x \cdot \psi_i$ and ψ_i' is immediate, since $\exists x \cdot \psi_i$ is more discriminating. For the other inclusion $[\psi_i'] \subseteq [\exists x \cdot \psi_i]$, let $\bar{a}' \in [\psi_i']$. Let $f_{\bar{a}'}$ be the natural embedding of $\mathcal{A}_{\psi_i'}$ into $\mathbb{A}$ mapping each equivalence class of variables in $\mathcal{A}_{\psi_i'}$ to the corresponding element in $\bar{a}'$. Similarly, since $\mathcal{A}_{\psi_i} \sqsubseteq \mathbb{A}$, there exists a tuple $\bar{a}b$ and an embedding $g_{\bar{a}b}$ of $\mathcal{A}_{\psi_i}$ into $\mathbb{A}$, where $g_{\bar{a}b}([x]) = b$. The substructure induced by $\bar{a}$ is isomorphic to that induced by $\bar{a}'$. Let h be such an isomorphism. Since $\mathbb{A}$ is homogeneous, h extends to a full automorphism of $\mathbb{A}$. Define $b' = h(b)$. Then, $\bar{a}'b' \in [\psi_i]$, and thus $\bar{a}' \in [\exists x \cdot \psi_i]$.

The universal quantifier is handled with the equivalence $\forall x \cdot \varphi \equiv \neg\exists x \cdot \neg\varphi$: First we replace $\neg\varphi$ by an equivalent formula in ldnf ψ by applying Proposition 3. Then, we apply the procedure above to remove the existential quantifier in $\exists x \cdot \psi$, and we thus obtain another formula ψ' in ldnf s.t. $\exists x \cdot \neg\varphi \equiv \psi'$. Finally, a further application of Proposition 3 to $\neg\psi'$ yields a formula ψ'' in ldnf s.t. $\psi'' \equiv \neg\exists x \cdot \neg\varphi$. ◀

By repeatedly using Proposition 4, we can implement the saturation algorithm in T-relative pseudo-polynomial time: First, transform all the formulas defining states and transitions of the input automata $\mathcal{P}$ and $\mathcal{A}$ into ldnf. Then, in every iteration, the formula forced(δ') is also transformed into ldnf. Step (2) is implemented by computing the union of clauses, and the implication in step (3) reduces to the inclusion of the sets of clauses of forced(δ') into those of δ'. Thus, one iteration of the algorithm requires relative pseudo-polynomial time. The total number of iterations is bounded by the number of orbits of the set $Q \times \Gamma \times Q$, since in every iteration at least one orbit is added to δ'. By Proposition 1, the number of orbits in bounded by $2^{\mathrm{poly}(n)}$ where n is the dimension of $Q \times \Gamma \times Q$. Therefore, the concrete saturation algorithm runs in T-relative pseudo-polynomial time for homogeneous atoms. ◀

As a consequence of Theorem 7, under a bound on the width of input automata, the PDS reachability problem is in `PTime`, independently of the complexity $T(k)$ of the induced substructure problem. Moreover, the proof of Theorem 7 reveals that the polynomial above does not depend on the bound on width[8].

▶ **Corollary 8.** *The PDS reachability problem is fixed-parameter* `PTime`*, with the width of the input automaton as the parameter.*

In Theorem 7 we have shown that the complexity of the saturation procedure/reachability can be upper-bounded once we have a bound on the complexity of the induced substructure problem. We show below that, depending on the homogeneous structure, the latter problem (and thus reachability) can be of arbitrarily high complexity, or even undecidable. Therefore, the bound on the time complexity of induced substructure problem in Theorem 7 is a necessary assumption.

[8] We are grateful to Mikołaj Bojańczyk for noticing this fact.

▶ **Theorem 9.** *Let $X \subseteq \mathbb{N}$ be a set of natural numbers. There exists a homogeneous structure $\mathbb{A}_X$ s.t. membership in X is many-one reducible to the induced substructure problem for $\mathbb{A}_X$.*

Proof. Let $X \subseteq \mathbb{N}$ be an arbitrary set of natural numbers. Intuitively, we effectively encode the set of natural numbers in an infinite antichain of finite tournaments, and we construct a homogeneous structure $\mathbb{A}_X$ s.t., for every natural number $n \in \mathbb{N}$, $n \in X$ if, and only if, the encoding of n is an induced substructure of $\mathbb{A}_X$. We use the instantiation of the embedding partial order $\sqsubseteq$ to finite directed graphs: $G \sqsubseteq H$ if G is isomorphic to an induced subgraph of H. A *tournament* is a directed graph $T = (V, E)$ s.t., for every pair of vertices $x, y \in V$, either $(x, y) \in E$, or $(y, x) \in E$, but not both. It is known that there exists a countably infinite $\sqsubseteq$-antichain $\mathcal{T}$ of finite tournaments [21]. Let f be an efficiently computable bijective mapping between natural numbers and tournaments in the antichain $\mathcal{T}$. Let $\mathcal{T}_X$ be those finite tournaments T in $\mathcal{T}$ with $T = f(n)$ for some $n \in X$. The construction of $\mathbb{A}_X$ uses the following result.

▶ **Proposition 5** ([24]; see also [21])**.** *For every $\sqsubseteq$-upward-closed family $\mathcal{T}$ of finite tournaments, there is a homogeneous directed graph $\mathbb{A}$ such that, for every finite tournament T, $T \sqsubseteq \mathbb{A}$ if, and only if, $T \in \mathcal{T}$.*

Let $\mathbb{A}_X$ be the homogeneous directed graph obtained by applying the proposition above to the upward closure of the antichain $\mathcal{T}_X$. Then, for a natural number $n \in \mathbb{N}$, we have $n \in X$ if, and only if, the finite tournament $f(n)$ is in $\mathcal{T}_X$, which is the same as $f(n)$ being in the upward-closure of $\mathcal{T}_X$, since $f(n)$ is by construction in the antichain $\mathcal{T}$. By the proposition above, the latter property is equivalent to ask whether $f(n) \sqsubseteq \mathbb{A}_X$. Therefore, we can reduce membership in X to the induced substructure problem in $\mathbb{A}_X$. ◀

6 Examples of homogeneous structures

The purpose of this section is to provide concrete examples of homogeneous structures for which we can efficiently solve the reachability problem of FO-definable PDS. Those are well known in the model-theoretic community (cf. [24]), and we present them here in order to show the wide applicability of our results. We also present a general technique, called *wreath product*, which can be used to derive new homogeneous structures from known ones. Recall that, by Theorem 7, if $T(k)$ is the time complexity of the induced substructure problem of a homogeneous structure $\mathbb{A}$, then reachability of FO-definable PDS over $\mathbb{A}$ is decidable in T-relative pseudo-polynomial time. When the former problem is in `PTime`, reachability can be solved in `ExpTime` by the following corollary of Theorem 7.

▶ **Corollary 10.** *Let $\mathbb{A}$ be a homogeneous relational structure with a* `PTime` *induced substructure problem. For FO-definable PDS $\mathcal{P}$ over $\mathbb{A}$ and an FO-definable NFA $\mathcal{A}$ recognizing a regular set of configurations $L_{\mathcal{P}}(\mathcal{A})$, one can construct in* `ExpTime` *an FO-definable NFA $\mathcal{B}$ recognizing $L_{\mathcal{P}}(\mathcal{B}) = Reach_{\mathcal{P}}^{-1}(L_{\mathcal{P}}(\mathcal{A}))$. In particular, the FO-definable PDS reachability problem over $\mathbb{A}$ is in* `ExpTime`*.*

All the concrete examples that we provide in the sequel, and all infinitely many examples that can be obtained by applying the wreath product, have a `PTime` induced substructure problem, and thus reachability is in `ExpTime`.

Equality. Equality atoms $(\mathbb{D}, =)$ consist of a countably-infinite set $\mathbb{D}$ together with the equality relation. Automorphisms are permutations of $\mathbb{D}$. Homogeneity follows from the fact that any finite partial bijection $\mathbb{D} \to \mathbb{D}$ can be extended to a permutation of the whole set $\mathbb{D}$.

This is arguably the simplest homogeneous structure. The induced substructure problem is in PTime, since it amounts to check whether the interpretation of $=$ in a given finite structure is the equality relation. By Corollary 10, reachability for FO-definable PDS over equality atoms is in ExpTime. This subsumes the result of [26], which considers a special case of our model where, among other restrictions, the input and stack alphabets are 1-dimensional, and the transition relation is quantifier-free definable (instead of FO-definable). Additionally, [26] shows that the problem is ExpTime-hard for equality atoms.

All the examples below generalize equality atoms by adding more relations to the vocabulary. We omit equality, which is assumed to always be in the vocabulary.

Equivalence. *Equivalence atoms* $(\mathbb{D}, R)$ consist of a countably-infinite set $\mathbb{D}$ and an infinite-index equivalence relation R over $\mathbb{D}$ s.t. each one of the infinitely-many equivalence classes is itself an infinite subset of $\mathbb{D}$. An automorphism of equivalence atoms is a bijection f of $\mathbb{D}$ which respects R, in the sense that, for every $x, y \in \mathbb{D}$, $(x, y) \in R$ if, and only if, $(f(x), f(y)) \in R$. Equivalence atoms are homogeneous. (We will see later that equivalence atoms are isomorphic with the wreath product of equality atoms with itself.) This can model hierarchically nested data, where one can check whether two elements belong to the same equivalence class, and, if so, whether they actually are the same element. Higher nested equivalence atoms can be obtained by iterating this process: 0-nested equivalence atoms are just equality atoms; and for any $k \geq 0$, $(k+1)$-nested equivalence atoms can be seen as the disjoint union of infinitely many copies of k-nested equivalence atoms, with one additional equivalence relation that relates a pair of elements iff they belong to the same copy.

Total, betweenness, and cyclic order. *Total order atoms* $(\mathbb{Q}, \leq)$ can be presented as the rational numbers $\mathbb{Q}$ together with the natural total order $\leq$. Automorphisms are monotonic bijections of rational numbers. Homogeneity follows from the fact that $\leq$ is dense: A monotonic bijection $f : X \to Y$ over a finite domain X extends to an automorphism of $\mathbb{Q}$. The induced substructure problem is in PTime, since it amounts to check whether the interpretation of $\leq$ in a given finite structure is a total order. This can be used to model qualitative time, where events are totally ordered, but no information is available on the distance between them. Another instance is given by data-centric applications [16].

Betweenness order atoms $(\mathbb{Q}, B)$ use the betweenness relation B, which is obtained by considering the order $\leq$ up to reversal: $B(x, y, z)$ holds when x lies between y and z, i.e., either $y < x < z$ or $z < x < y$. This can be used to model time where one is not interested on the order between the events themselves, but rather on whether an event happened between two other events. *Cyclic order atoms* $(\mathbb{Q}, K)$ use the ternary cyclic ordering K obtained by bending the total order into a circle. Formally, $K(x, y, z)$ if either $x < y < z$, or $z < x < y$, or $y < z < x$. This can model a notion of qualitative cyclic time, where events cyclically repeat, but no precise timing information is available. For both betweenness and cyclic order atoms, the induced substructure problem is in PTime.

Universal partial order and preorder. Every relational homogeneous structure is obtained as the *Fraïssé limit* of the set of all its finite induced substructures [19]. (We do not formally define here the notion of Fraïssé limit, which is a central tool for constructing homogeneous structures; cf. [24].) For instance, total order atoms are the Fraïssé limit of all finite total orders. *Partial order atoms* are obtained as the Fraïssé limit of the set of all finite partial orders. The induced substructure problem amounts to determine whether the interpretation of $\leq$ in a given finite structure is a partial order, which can clearly be done in PTime. This

can be used to model the ordering of events in distributed systems. Along the same lines one obtains *preorder atoms*.

Universal tree order. A *tree order* (or semilinear order) is a partially ordered structure $(A, \leq)$ s.t. a) every two elements have an common upper bound, and b) for every element, its upward closure is totally ordered. Tree order atoms $(T, \leq)$ are obtained as the Fraissé limit of the set of all finite tree orders. Intuitively, tree order atoms consists of a countably-infinite tree order where each maximal path is isomorphic to total order atoms. Tree order atoms as presented here are not homogeneous. Intuitively, this happens because isomorphic substructures have least upper bounds outside the structures themselves, and they might relate to those in an incomparable way. This can be amended by introducing be the following ternary relation: $R(x, y, z)$ holds when the lub of x and y is incomparable with z. Then, $(T, \leq, R)$ is homogeneous, and it can be obtained as the Fraissé limit of the set of all extended finite tree orders $(A, \leq, R)$. The induced substructure problem is in `PTime` for $(T, \leq, R)$.

Universal graph and tournament. *Universal graph atoms* are obtained as the Fraissé limit of the set of all finite graphs. This is also known as *Rado's graph* or *the random graph*. The induced substructure problem is trivial since the universal graph contains an isomorphic copy of every finite graph. Similarly, *universal tournament atoms* are the Fraissé limit of the set of all finite tournaments, where a tournament is an irreflexive graph $T = (V, E)$ s.t., for every two nodes $x, y \in V$, either $(x, y) \in E$, or $(y, x) \in E$. Given a graph, it is clearly checkable in `PTime` whether it is actually a tournament, thus the induced substructure problem is in `PTime` also in this case.

Wreath products. We conclude this section by giving a construction which allows to compose homogeneous structures in order to produce new ones. Given two relational structures $\mathbb{A} = (A, R_1, \ldots, R_m)$ and $\mathbb{B} = (B, S_1, \ldots, S_n)$, their *wreath product* is the relational structure $\mathbb{A} \otimes \mathbb{B} = (A \times B, R'_1, \ldots, R'_m, S'_1, \ldots, S'_n)$, where $((a_1, b_1), \ldots, (a_k, b_k)) \in R'_i$ if $(a_1, \ldots, a_k) \in R_i$, and $((a_1, b_1), \ldots, (a_k, b_k)) \in S'_j$ if $a_1 = \cdots = a_k$ and $(b_1, \ldots, b_k) \in S_j$. Intuitively, $\mathbb{A} \otimes \mathbb{B}$ is obtained by replacing each element in $\mathbb{A}$ with a disjoint copy of $\mathbb{B}$. It can be checked that, if the two structures $\mathbb{A}$ and $\mathbb{B}$ are homogeneous, then the same holds for their wreath product $\mathbb{A} \otimes \mathbb{B}$. The induced substructure problem for $\mathbb{A} \otimes \mathbb{B}$ reduces in `PTime` to the same problem for $\mathbb{A}$ and $\mathbb{B}$: $\{(a_1, b_1), \ldots, (a_k, b_k)\}$ is an induced substructure of $\mathbb{A} \otimes \mathbb{B}$ if, and only if, $\{a_1, \ldots, a_k\}$ is an induced substructure of $\mathbb{A}$, and for every i, $\{b_j \mid a_j = a_i\}$ is an induced substructure of $\mathbb{B}$. Therefore, if both $\mathbb{A}$ and $\mathbb{B}$ have a `PTime` induced substructure problem, then the same holds for $\mathbb{A} \otimes \mathbb{B}$, and Corollary 10 applies.

As an application of the wreath product, take $\mathbb{A}_0 = (\mathbb{D}, =)$ to be equality atoms, and, for each $k \geq 0$, let $\mathbb{A}_{k+1} = \mathbb{A}_0 \otimes \mathbb{A}_k$. Then, $\mathbb{A}_1$ is just the equivalence atoms presented before, and, more generally, $\mathbb{A}_k = (\mathbb{D}, R_1, \ldots, R_k)$ is *k-nested equivalence atoms*, which can be used to model data with nested equivalence relations. For each of those infinitely many examples, the reachability problem for FO-definable PDS is in `ExpTime`.

7 Conclusions

We have studied the reachability problem for a model of PDS with countably-infinite FO-definable states, stack alphabet, and transitions relation. We advocate a Ockham's razor research strategy that refrains from inventing seemingly new notions. Instead, we have taken the standard definition of PDS and re-interpreted it in the richer framework of FO-definable

sets instead of ordinary finite sets. This covers the well-known model of pushdown register automata [12, 26] as one instantiation of the general paradigm, and we have shown that the optimal `ExpTime` complexity for the reachability problem for this model can be recovered in the more general framework. This same paradigm can of course be applied to a variety of different models, like timed PDS [1], data/timed extensions of Petri nets [3, 23], lossy channel systems [2], 1-clock/1-register alternating automata [22, 27, 15], rewriting systems [8], etc. Therefore, the present paper can be seen as a proof of concept of the new research strategy. For example, one could consider *FO-definable pushdown automata* (PDA) and *FO-definable context-free grammars* (CFG) as acceptors of languages over infinite alphabets. The definition of FO-definable PDA is analogous to PDS, except that the transition relation is an FO-definable subset of $Q \times \Gamma^* \times A_\varepsilon \times Q \times \Gamma^*$, where $A_\varepsilon = A \cup \{\varepsilon\}$ is an FO-definable alphabet extended with the empty word. Similarly, FO-definable CFG can be defined as *stateless* FO-definable PDA where every transition pops exactly one symbol from the stack. It is easy to prove that FO-definable PDA languages coincide with FO-definable context-free languages for oligomorphic atoms [5], and that the latter are closed under union, concatenation, Kleene star, homomorphism, inverse homomorphism, intersection with FO-definable regular languages, and that collapsing each orbit to a different symbol yields a classical context-free language.

References

1 P. A. Abdulla, M. F. Atig, and J. Stenman. Dense-timed pushdown automata. In *Proc. of LICS'12*, pages 35–44, june 2012.

2 Parosh Aziz Abdulla, Mohamed Faouzi Atig, and Jonathan Cederberg. Timed lossy channel systems. In *Proc. of FSTTCS'12*, volume 18 of *LIPIcs*, pages 374–386, 2012.

3 Parosh Aziz Abdulla and Aletta Nylén. Timed Petri nets and BQOs. In *Proc. of ICATPN'01*, pages 53–70, 2001.

4 Mohamed F. Atig. Model-checking of ordered multi-pushdown automata. *Log. Methods Comput. Sci.*, 8(3), 09 2012.

5 Mikołaj Bojańczyk, Bartek Klin, and Sławomir Lasota. Automata theory in nominal sets. *Logical Methods in Computer Science*, 10(3:4):paper 4, 2014.

6 Benedikt Bollig, Aiswarya Cyriac, Paul Gastin, and K. Narayan Kumar. Model checking languages of data words. In Lars Birkedal, editor, *Proc. of FOSSACS'12*, volume 7213 of *LNCS*, pages 391–405. Springer, 2012.

7 Ahmed Bouajjani, Javier Esparza, and Oded Maler. Reachability analysis of pushdown automata: Application to model-checking and saturation method. In *Proc. of CONCUR'97*, volume 1243 of *LNCS*, pages 135–150, 1997.

8 Ahmed Bouajjani, Peter Habermehl, Yan Jurski, and Mihaela Sighireanu. Rewriting systems with data. In *In Proc. of FCT'07*, volume 4639 of *LNCS*, pages 1–22. Springer, 2007.

9 Luca Breveglieri, Alessandra Cherubini, Claudio Citrini, and Stefano Crespi-Reghizzi. Multi-push-down languages and grammars. *Int. J. Found. Comput. Sci.*, 7(3):253–292, 1996.

10 Chris Broadbent, Arnaud Carayol, Matthew Hague, and Olivier Serre. A saturation method for collapsible pushdown systems. In *Proc. of ICALP'12*, volume 7392 of *LNCS*, pages 165–176, 2012.

11 Arnaud Carayol and Matthew Hague. Saturation algorithms for model-checking pushdown systems. In *Proc. of AFL'14*, volume 151 of *EPTCS*, pages 1–24, 5 2014.

12 Edward Y. C. Cheng and Michael Kaminski. Context-free languages over infinite alphabets. *Acta Inf.*, 35(3):245–267, 1998.

13 L. Clemente and S. Lasota. Timed pushdown automata revisited. In *Proc. of LICS'15*, 2015. Accepted for publication.

14 Wojciech Czerwiński, Piotr Hofman, and Sławomir Lasota. Reachability problem for weak multi-pushdown automata. *Logical Methods in Computer Science*, 9(3:13):1–29, 2013.

15 Stéphane Demri and Ranko Lazic. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Logic*, 10(3):16:1–16:30, April 2009.

16 Alin Deutsch, Richard Hull, Fabio Patrizi, and Victor Vianu. Automatic verification of data-centric business processes. In *Proc. of ICDT'09*, pages 252–267, New York, NY, USA, 2009. ACM.

17 Javier Esparza and Stefan Schwoon. A BDD-based model checker for recursive programs. In *Proc. of CAV'01*, CAV'01, pages 324–336. Springer-Verlag, 2001.

18 Alain Finkel, Bernard Willems, and Pierre Wolper. A direct symbolic approach to model checking pushdown systems. In *Proc. of INFINITY'97*, volume 9, pages 27–37, 1997.

19 R. Fraïssé. *Theory of relations*. North-Holland, 1953.

20 Irène Guessarian. Pushdown tree automata. *Theor. Comp. Sys.*, 16:237–263, 1983.

21 Ward Henson. Countable homogeneous relational structures and $\aleph_0$-categorical theories. *J. Symb. Logic*, 37:494–500, 1972.

22 Slawomir Lasota and Igor Walukiewicz. Alternating timed automata. *ACM Trans. Comput. Logic*, 9(2):10:1–10:27, 2008.

23 Ranko Lazic, Tom Newcomb, Joël Ouaknine, A. W. Roscoe, and James Worrell. Nets with tokens which carry data. In *Proc. of ICATPN'07*, LNCS, pages 301–320. Springer-Verlag, 2007.

24 Dugald Macpherson. A survey of homogeneous structures. *Discrete Mathematics*, 311(15):1599–1634, 2011.

25 A. N. Maslov. Multilevel stack automata. *Probl. Peredachi Inf.*, 12(1):55–62, 1976.

26 Andrzej S. Murawski, Steven J. Ramsay, and Nikos Tzevelekos. Reachability in pushdown register automata. In *MFCS 2014*, pages 464–473, 2014.

27 Joel Ouaknine and James Worrell. On the language inclusion problem for timed automata: Closing a decidability gap. In *In Proc. of LICS'04*, pages 54–63. IEEE Computer Society, 2004.

Relational Semantics of Linear Logic and Higher-order Model Checking

Charles Grellois[1] **and Paul-André Melliès**[2]

1 **Laboratoires PPS & LIAFA, Université Paris Diderot, France**
grellois@pps.univ-paris-diderot.fr
2 **Laboratoire PPS, CNRS & Université Paris Diderot, France**
mellies@pps.univ-paris-diderot.fr

Abstract

In this article, we develop a new and somewhat unexpected connection between higher-order model-checking and linear logic. Our starting point is the observation that once embedded in the relational semantics of linear logic, the Church encoding of a higher-order recursion scheme (HORS) comes together with a dual Church encoding of an alternating tree automata (ATA) of the same signature. Moreover, the interaction between the relational interpretations of the HORS and of the ATA identifies the set of initial states of the tree automaton from which the infinite tree generated by the recursion scheme is accepted. We show how to extend this result to alternating parity automata (APT) by introducing a parametric version of the exponential modality of linear logic, capturing the formal properties of colors (or priorities) in higher-order model-checking. We show in particular how to reunderstand in this way the type-theoretic approach to higher-order model-checking developed by Kobayashi and Ong. We briefly explain in the end of the paper how this analysis driven by linear logic results in a new and purely semantic proof of decidability of the formulas of the monadic second-order logic for higher-order recursion schemes.

1998 ACM Subject Classification D.2.4 Software/Program Verification, F.3.2 Semantics of Programming Languages, F.4.1 Mathematical Logic

Keywords and phrases Higher-order model-checking, linear logic, relational semantics, parity games, parametric comonads

Digital Object Identifier 10.4230/LIPIcs.CSL.2015.260

1 Introduction

Thanks to the seminal works by Girard and Reynolds on polymorphism and parametricity in the 1970s, it has been recognized that every finite tree t on a given signature Σ can be seen alternatively as a simply-typed λ-term of an appropriate type depending on Σ. This correspondence between trees and λ-terms is even bijective if one considers λ-terms up to $\beta\eta$-equivalence, see for instance Girard [8]. Typically, a finite tree t on the signature

$$\Sigma \quad = \quad \{\, a : 2 \,,\, b : 1 \,,\, c : 0 \,\} \tag{1}$$

is the same thing under this Church encoding as a simply-typed λ-term t of type

$$(o \to o \to o) \to (o \to o) \to o \to o \tag{2}$$

modulo $\beta\eta$-equivalence. The idea underlying the correspondence is that every constructor a, b, c of the signature Σ should be treated as a variable

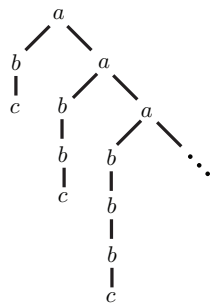
$$a \;:\; o \to o \to o \qquad b \;:\; o \to o \qquad c \;:\; o \tag{3}$$

24th EACSL Annual Conference on Computer Science Logic (CSL 2015).
Editor: Stephan Kreutzer; pp. 260–276
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

where the number of inputs o in the type $o \to \cdots \to o \to o$ of the variable a, b, c indicates the arity of the combinator. An equally well-known fact is that this translation extends to infinite trees generated by higher-order recursion schemes on the signature Σ if one extends the simply-typed λ-calculus with a fixpoint operator Y. For example, the higher-order recursion scheme $\mathcal{G}$ on the signature Σ

$$\mathcal{G} \;=\; \begin{cases} S & \mapsto \;\; F\ a\ b\ c \\ F\ x\ y\ z & \mapsto \;\; x\ (y\ z)\ (F\ x\ y\ (y\ z)) \end{cases} \tag{4}$$

constructs the infinite tree

$[\![\mathcal{G}]\!] \quad =$ 

(5)

and can be formulated as a λ-term of the same type (3) as previously but defined in the simply-typed λ-calculus extended with the fixpoint operator Y:

$$\lambda abc.\ ((Y\ \ (\lambda F.\, (\lambda xyz.\, x\, (y\, z)\ \ (F\, x\, y\, (y\, z))))))\ \ a\, b\, c) \tag{6}$$

A natural temptation is to study the correspondence between higher-order recursion schemes (4) and simply-typed λ-terms with fixpoints (6) from the resource-aware point of view of linear logic. Recall that the intuitionistic type (2) is traditionally translated in linear logic as the formula

$$A \quad = \quad !(\ !o \multimap !o \multimap o\ ) \multimap !(\ !o \multimap o\ ) \multimap !o \multimap o.$$

As expected, the higher-order recursion scheme $\mathcal{G}$ in (4) can be translated as a proof t_A of this formula A in linear logic extended with a fixpoint operator Y. An amusing and slightly puzzling observation is that the scheme $\mathcal{G}$ can be alternatively translated as a proof t_B of the formula B below:

$$B \quad = \quad !(\ o \multimap o \multimap o\ ) \multimap !(\ o \multimap o\ ) \multimap !o \multimap o$$

with the same underlying simply-typed λ-term with fixpoint operator Y. The difference between the terms t_A and t_B is not syntactic, but type-theoretic: in the case of the term t_A, the type A indicates that each tree-constructor a, b and c of the signature Σ is allowed to call its hypothesis as many times as desired:

$$a \;:\; !o \multimap !o \multimap o \qquad b \;:\; !o \multimap o \qquad c \;:\; o$$

whereas in the case of the term t_B, the type B indicates that each variable a, b, c calls each of its hypothesis exactly once:

$$a \;:\; o \multimap o \multimap o \qquad b \;:\; o \multimap o \qquad c \;:\; o$$

As a matter of fact, it appears that the proof t_B is the image of the proof t_A along a canonical coercion of linear logic

$$\vdash \iota : A \multimap B.$$

The status of this program transformation ι is difficult to understand unless one recalls that linear logic is based on the existence of a perfect duality between the programs of a given type A and their environments or counter-programs which are typed by the linear negation $A^\perp$ of the original type A. Accordingly, since the two terms t_A and $t_B = \iota \circ t_A$ are syntactically equal, their difference should lie in the class of counter-programs of type $A^\perp$ or $B^\perp$ which are allowed to interact with them. This idea takes its full flavour in the context of model-checking, when one realizes that every tree automaton $\mathcal{A}$ on the signature Σ may be seen as a counter-program whose purpose is indeed to interact with t_A or t_B in order to check whether a specific property of interest is satisfied by the infinite tree $[\![\mathcal{G}]\!]$ generated by the recursion scheme $\mathcal{G}$. This leads to the tentative duality principle:

$$\begin{array}{c}\text{higher-order}\\ \text{recursion schemes } \mathcal{G}\end{array} \qquad \leftrightsquigarrow \qquad \text{tree automata } \mathcal{A}$$

where a tree automaton $\mathcal{A}$ on the signature Σ is thus seen as a counter-program of type $A^\perp$ or $B^\perp$ interacting with the higher-order recursion scheme $\mathcal{G}$ seen as a program of type A or B. An apparent obstruction to this duality principle is that, in contrast to what happens with recursion schemes $\mathcal{G}$, it is in general impossible to translate a tree automaton $\mathcal{A}$ as a proof of linear logic — in particular because linear logic lacks non-determinism. However, one neat way to resolve this matter and to extend linear logic with non-determinism is to embed the logic in its relational semantics, based on the monoidal category *Rel* of sets and relations. The relational semantics of linear logic is indeed entitled to be seen as a non-deterministic extension of linear logic where every nondeterministic tree automaton $\mathcal{A} = \langle \Sigma,\, Q,\, \delta, q_0 \rangle$ may be "implemented" by interpreting the base type o as the set Q of states of the automaton, and by interpreting each variable a, b, c as the following relations

$$a \;:\; Q \multimap Q \multimap Q \qquad b \;:\; Q \multimap Q \qquad c \;:\; Q$$

deduced from the transition function δ of the automaton:

$$\begin{gathered} a = \{(q_1, q_2, q) \in Q \times Q \times Q \mid (1, q_1) \wedge (2, q_2) \in \delta(q, a)\} \\ b = \{(q_1, q) \in Q \times Q \mid (1, q_1) \in \delta(q, b)\} \\ c = \{q \in Q \mid \delta(q, c) = \mathit{true}\} \end{gathered}$$

The nondeterministic tree automaton $\mathcal{A}$ is then interpreted as the counter-program $\mathcal{A}_B = !\,a \otimes !\,b \otimes !\,c \otimes d$ of type

$$B^\perp \;=\; !\,(Q \multimap Q \multimap Q) \otimes !\,(Q \multimap Q) \otimes !\,Q \;\otimes\; Q^\perp.$$

obtained by tensoring the three relations a, b, c lifted with by the exponential modality $!$ together with the singleton $d = \{q_0\}$ consisting of the initial state of the automaton, and understood as a counter-program of type $Q^\perp$. Note that by composition with the contraposite $\iota^\perp : B^\perp \multimap A^\perp$ of the coercion ι, one gets a counter-program $\mathcal{A}_A = \iota^\perp \circ \mathcal{A}_B$ of type

$$A^\perp \;=\; !\,(!\,Q \multimap !\,Q \multimap Q) \otimes !\,(!\,Q \multimap Q) \otimes !\,Q \;\otimes\; Q^\perp.$$

Note also that when the type o is interpreted as Q in the relational model, the counter-programs of type $B^\perp$ of the form $!\,a \otimes !\,b \otimes !\,c \otimes d$ with $d = \{q_0\}$ correspond exactly to the non-deterministic tree automata on the signature Σ with set of states Q and initial state q_0. The difference between the two types A and B becomes very clear and meaningful at this stage: shifting to the type $A^\perp$ enables one to extend the class of nondeterministic

tree automata to nondeterministic *alternating* tree automata $\mathcal{A}$ with typical transitions of the form

$$\delta(q,a) \quad = \quad (1,q_1) \ \wedge \ (1,q_2) \tag{7}$$

meaning that the tree automaton $\mathcal{A}$ meeting the tree $a(t_1,t_2)$ at state q explores the left subtree t_1 *twice* with states q_1 and q_2 and does not explore *at all* the right subtree t_2. Such a transition $\delta(q,a)$ is typically represented in the relational semantics of linear logic by the singleton relation

$$a \ = \ \{ \ (\ \{\!|q_1,q_2|\!\} \, , \, \emptyset \, , \, q \ ) \ \} \ : \ !Q \multimap !Q \multimap Q \tag{8}$$

where one uses the set $!Q$ of finite multisets of Q to encode the transition (7) with the finite multiset $\{\!|q_1,q_2|\!\}$ consisting of the two states $q_1,q_2 \in Q$ and the empty multiset $\emptyset$ of states. It should be stressed that a tree automaton $\mathcal{A}$ admitting such an "alternating" transition $\delta(q,a)$ *cannot* be encoded as a counter-program of type $B^\perp$ because the transitions of the tree automaton $\mathcal{A}$ are *linear* in that type and thus explore *exactly* once each subtree t_1 and t_2 of the tree $a(t_1,t_2)$. Summarizing the current discussion, we are entitled to consider that each linear type $A^\perp$ and $B^\perp$ reflects a specific class of tree automata on the signature Σ:

$$\begin{array}{ccc} B^\perp & \leftrightarrow & \text{non-deterministic tree automata} \\ A^\perp & \leftrightarrow & \text{non-deterministic } \textit{alternating} \text{ tree automata} \end{array}$$

Accordingly, the purpose of the coercion ι from t_A to t_B is to restrict the power of the class of alternating non-deterministic tree automata allowed to explore the infinite tree $[\![\mathcal{G}]\!]$ generated by the higher-order recursion scheme $\mathcal{G}$ of signature Σ.

Description of the paper. The purpose of this paper is to show that this duality between recursion schemes and tree automata underlies several of the recent developments in the field of higher-order model-checking. To that purpose, after recalling in §2 the notion of alternating parity tree automaton and of monadic second-order logic over trees, we start by establishing in §3 an equivalence between the intersection type system introduced by Kobayashi to describe infinitary coinductive proofs, and an infinitary variant of the traditional relational semantics of linear logic developed in [10]. This correspondence between an intersection type system and a relational semantics of linear logic adapts to the field of higher-order model-checking ideas dating back to Coppo, Dezani, Honsell and Longo [4] and recently revisited by Terui [27] and independently by de Carvalho [5] in order to establish complexity properties of evaluation in the simply-typed λ-calculus. It may be also seen as an account based on linear logic of the semantic approach to higher-order model-checking developed by Aehlig in the early days of the field [1]. The main contribution of the paper is the observation developed in §4 that this correspondence between intersection type systems and the relational semantics of linear logic extends to alternating parity games, and thus to the full hierarchy of the modal μ-calculus. This extension relies on the construction of a *parametric comonad* in the sense of [18] defined as a family of modalities $\Box_m$ indexed by colors (or priorities) $m \in \mathbb{N}$, equipped with a series of structural morphisms satisfying suitable coherence properties. The resulting intersection type system provides a clean and conceptual explanation for the type system designed by Kobayashi and Ong [17] in order to accomodate the hierarchy of colors. In particular, we show that a simpler but equivalent treatment of colors is possible. Finally, we explain in §4 in what sense the parametric comonad exhibited at the level of intersection types corresponds to a traditional notion of exponential modality at the level of the relational semantics of linear logic. We obtain in this way a semantic reformulation of alternating

parity games in an infinitary and colored variant of the relational semantics of linear logic. In particular, just as for finitary tree automata, there exists an accepting run-tree from an initial state $q \in Q$ of the alternating parity automaton if and only if the state q is an element of the composite (in the relational semantics) of the recursion scheme with the tree automaton.

2 Logical specification and automata theory

2.1 Monadic second-order logic and modal μ-calculus

The purpose of higher-order model-checking is to abstract the behavior of a functional program with recursion as a tree approximating the set of its potential executions, and then to specify a logical property to check over this tree. The tradition in higher-order model-checking is to consider monadic second-order logic, a well-balanced choice between expressivity – it contains most other usual logics over trees – and complexity: the satisfiability of a formula is decidable for infinite structures of interest – as infinite trees (Rabin 1969). Higher-order verification has a different approach: the question is whether a *given tree* satisfies the formula – or whether an equivalent automaton accepts it. A key step towards an automata-theoretic approach to MSO comes from the work by Janin and Walukiewicz[15]:

▶ **Theorem.** *MSO is equi-expressive to modal μ-calculus over trees.*

Recall that modal μ-calculus formulas are defined by the grammar

$$\phi \ ::= \ X \mid \underline{f} \mid \phi \vee \phi \mid \phi \wedge \phi \mid \Box\, \phi \mid \ \diamond_i \, \phi \mid \mu X.\, \phi \mid \nu X.\, \phi$$

for $f \in \Sigma$. Given a ranked tree, the semantics of a formula is the set of nodes where it holds. The predicate $\underline{f}$ is true on f-labelled nodes, $\Box\, \phi$ is true on nodes whose succesors all satisfy ϕ, $\diamond_i \, \phi$ is true on nodes whose i^{th} succesor satisfies ϕ, and the μ and ν are two fixpoints operators which can be understood in two different manners. Semantically, they are dual operators, μ and ν being respectively the least and greatest fixpoint on the semantics of formulas.

Given a Σ-labelled ranked tree whose set of nodes is N, whose branching structure is given by a finite family of successor functions $succ_i : N \to N$, and whose labelling is described by a function $label : N \to \Sigma$, the semantics of a closed modal μ-calculus formula ϕ is defined as $||\phi||_\emptyset$ where $\emptyset$ denotes the unique function $\emptyset \to N$, and for a function $\mathcal{V} : Var \to N$ and a modal μ-calculus formula ψ, the semantics $||\psi||_\mathcal{V}$ are defined inductively:

- $||a||_\mathcal{V} \ = \ \{n \in N \mid label(n) \ = \ a\}$
- $||X||_\mathcal{V} \ = \ \mathcal{V}(X)$
- $||\neg\phi||_\mathcal{V} \ = \ N \setminus ||\phi||_\mathcal{V}$
- $||\phi \vee \psi||_\mathcal{V} \ = \ ||\phi||_\mathcal{V} \cup ||\psi||_\mathcal{V}$
- $||\diamond_i \ \phi||_\mathcal{V} \ = \ \{n \in N \mid ar(n) \geq i \text{ and } succ_i(n) \in ||\phi||_\mathcal{V}\}$
- $||\mu X.\, \phi(X)||_\mathcal{V} \ = \ \bigcap \{M \subseteq N \mid ||\phi(X)||_{\mathcal{V}[X \leftarrow M]} \subseteq M\}$

where $\mathcal{V}[X \leftarrow M]$ coincides with $\mathcal{V}$ except on X which it maps to M. The semantics of $\wedge$, $\Box$ and ν are defined using de Morgan duality.

Another understanding of μ and ν is syntactic and closer to automata theory: both allow the unfolding of formulas

$$\mu X.\, \phi[X] \ \to_\mu \phi[\mu X.\, \phi[X]] \text{ and } \nu X.\, \phi[X] \ \to_\nu \phi[\nu X.\, \phi[X]]$$

but $\to_\mu$ may only be expanded finitely, while $\to_\nu$ is unconstrained. The semantics of a formula may then be understood as the set of positions from which it admits unfoldings which are logically true and which do not use $\to_\mu$ infinitely.

2.2 Alternating parity tree automata

From this syntactic interpretation of fixpoints over formulas, we can define a class of automata corresponding to modal μ-calculus, namely *alternating parity tree automata* (APT), whose purpose is to synchronise the unraveling of formulas with symbols of the tree. These automata are top-down tree automata, with two additional features:

- *alternation*: they have the power to duplicate or drop subtrees, and to run with a different state on every copy,
- and *parity conditions*: since run-trees are infinitary by nature, these automata discriminate *a posteriori* the run-tree unfolding $\to_\mu$ infinitely.

The transition function takes values in positive Boolean formulas over couples of states and directions, its generic shape being

$$\delta(q,\, a) \quad = \quad \bigvee_{i \in I} \bigwedge_{j \in J_i} (d_{i,j},\, q_{i,j}) \tag{9}$$

which consists of a non-deterministic choice of i followed by the execution of $|J_i|$ copies of the automaton, each on the successor in direction $d_{i,j}$ of the current node, with state $q_{i,j}$.

When for every $i \in I$ and every direction d there is a unique $j \in J_i$ such that $d_{i,j} = d$, we recover the usual notion of *non-deterministic* parity automaton. States of an APT may be understood as subformulas of the formula of interest, so that some correspond to subformulas $\mu X.\, \phi$ and others to subformulas $\nu X.\, \phi$. To exclude infinite unfoldings of μ, every state q is attributed a *color* $\Omega(q) \in \mathbb{N}$. States in the immediate scope of a μ receive an odd color, and the others an even one. If q corresponds to a subformula of q', then the coloring will satisfy $\Omega(q) \leq \Omega(q')$. The construction of Ω is such that the greatest color among the ones seen infinitely often in an infinite branch informs the automaton about which fixpoint operator was unfolded infinitely along it.

A branch of a run-tree is *winning* when the greatest color seen infinitely often along it is even. A run-tree is declared *winning* when all its infinite branches are. Emerson and Jutla [7] prove that every modal μ-calculus formula ϕ can be translated to an equivalent APT $\mathcal{A}_\phi$, in the sense that

▶ **Theorem.** *Given a Σ-labelled ranked tree T, ϕ holds at the root of T if and only if $\mathcal{A}_\phi$ has a winning run-tree over T.*

3 The type-theoretic approach to higher-order model-checking

Developing an idea by Hosoya, Pierce and Vouillon [14], Kobayashi designed in [16] a type-theoretic account of higher-order model-checking in the particular case of an alternating tree automata (ATA) testing for coinductive properties — and thus *without* parity conditions. In this section, we briefly recall his terminology and results, and explain the hidden connection with relational semantics.

3.1 Recursion schemes and simply-typed λ-terms with fixpoint

Given a ranked alphabet Σ, we will consider in this paper two kinds of Σ-labelled trees of finite or countable depth:

- *ranked* trees, typically generated by higher-order recursion schemes, in which the number of children of a node labelled with $f \in \Sigma$ is equal to the arity $\mathrm{ar}(f)$ of its label,
- *unranked* trees, typically used to describe run-trees of alternating automata, and where the previous arity constraint is relaxed.

Given a base type o, we will consider the set $\mathcal{K}$ of simple types, generated by the grammar

$$\kappa \quad ::= \quad o \mid \kappa \to \kappa$$

modulo associativity of the arrow to the right. Every simple type has a unique decomposition

$$\kappa \quad = \quad \kappa_1 \to \cdots \to \kappa_n \to o$$

where n is the *arity* of κ, denoted $\mathrm{ar}(\kappa)$. The complexity of κ is typically measured by its *order*, defined inductively by $\mathrm{order}(\kappa) = 0$ if $n = 0$ and

$$\mathrm{order}(\kappa) \;=\; 1 + \max(\mathrm{order}(\kappa_1), \ldots, \mathrm{order}(\kappa_n))$$

In the sequel, following Kobayashi [16], we shall refer to simple types as *kinds*, to prevent confusions with intersection types. We write $f :: \kappa$ or $t :: \kappa$ when a symbol f or a term t has kind κ. The formalism of *higher-order recursion schemes* (HORS) on a given ranked signature Σ may be seen as equivalent to simply-typed λ-calculus with a recursion operator Y and free variables $f \in \Sigma$ of order at most 1. Consequently, every free variable $f \in \Sigma$ of the λY-term corresponding to the recursion scheme has kind

$$\underbrace{o \;\to\; \cdots \;\to\; o}_{\mathrm{ar}(f)} \;\to\; o. \tag{10}$$

where $\mathrm{ar}(f)$ denotes the arity of the terminal $f \in \Sigma$. The normalization of the λY-term associated to the recursion scheme $\mathcal{G}$ produces a potentially infinite ranked tree, labelled by its free variables. As we explained in the introduction, this translation of higher-order recursion schemes into λY-terms may be seen as an instance of the Church encoding of ranked trees over the signature Σ.

In order to check whether a given monadic second-order formula holds at the root of the infinite tree generated by a HORS, a traditional procedure is to explore it using an *alternating parity automaton* (APT). Every exploration of the APT produces a run-tree labelled by the same signature, but unranked because of the alternating nature of the automaton. The subclass of APT in which every state of the automaton is assigned color 0 (the least coinductive priority) defines the class of *alternating tree automata* (ATA), which can test coinductive properties like safety, but cannot test inductive properties like reachability. The definitions of HORS and APT, as well as the correspondence between APT and monadic second-order logic, are recalled in the Appendix. In the sequel, S denotes the start symbol of a recursion scheme $\mathcal{G}$, $\mathcal{N}$ its set of non-terminals, and $\mathcal{R}(F)$ denotes, for every non-terminal $F \in \mathcal{N}$, the simply-typed λ-term it rewrites to as $F \to_{\mathcal{G}} \mathcal{R}(F)$ in the recursion scheme $\mathcal{G}$.

3.2 From kinds to intersection types

In his original work, Kobayashi reduces the study of coinductive properties of higher-order recursion schemes to the definition of an intersection type system. The general idea is that every transition of an alternating tree automaton of the form

$$\delta(q_0, \mathtt{if}) = (2, q_0) \wedge (2, q_1) \tag{11}$$

may be understood type-theoretically as a refinement of the simple type

$$\texttt{if} :: o \to o \to o$$

and reformulated as an *intersection type*

$$\emptyset \to (q_0 \wedge q_1) \to q_0.$$

This intersection type expresses the fact that, given any tree T_1 and a tree T_2 accepted from both states q_0 and q_1, the composed tree $\texttt{if}\ T_1\ T_2$ is accepted from the state q_0. Following this connection, Kobayashi defines for every HORS $\mathcal{G}$ and every alternating tree automaton $\mathcal{A}$ *without parity condition* a type system $Kob(\mathcal{G}, \mathcal{A})$ satisfying the following property:

▶ **Theorem 1** (Kobayashi [16]). *The sequent*

$$\vdash_{\mathcal{G},\mathcal{A}} \ S : q :: o$$

is provable in $Kob(\mathcal{G}, \mathcal{A})$ if and only if there is a run-tree of $\mathcal{A}$ over $[\![\mathcal{G}]\!]$ with initial state q.

Note that the intersection type system $Kob(\mathcal{G}, \mathcal{A})$ is somewhat ad hoc since it depends on $\mathcal{G}$ and $\mathcal{A}$, in contrast to the approach developed in the present paper, based on a Church encoding of $\mathcal{G}$ and $\mathcal{A}$ in a single intersection type system formulated in §4 and §5.

3.3 Intersection types and the relational semantics of linear logic

As a warm-up to the next two sections §4 and §5, and to the modal treatment of colors in alternating parity tree automata (APT), we explain here in the simpler coinductive case, how to relate Kobayashi's intersection type system for alternating tree automata (ATA) to an infinitary variant of the relational semantics of linear logic. As already explained, the Church encoding of a ranked tree over the signature $\Sigma = \{f_i : ar_i \,|\, i \in I\}$ defines a λY-term t of simple type o with free variables f_i of kind (10), translated as the following formula of linear logic:

$$f_i \quad : \quad \underbrace{!\,o \multimap \cdots \multimap !\,o}_{ar_i} \multimap o \qquad \text{for } i \in I.$$

The λY-term t itself is thus typed by the following sequent of linear logic:

$$\cdots \quad , \quad f_i \quad : \quad !\Big(\underbrace{!\,o \multimap \cdots \multimap !\,o}_{a_i} \multimap o \Big) \quad , \quad \cdots \quad \vdash \quad t \quad : \quad o$$

From this follows that its interpretation $[\![t]\!]$ in the relational semantics of linear logic defines a subset of the following set of "higher-order states"

$$[\![t]\!] \ \subseteq \ \left[\!\!\left[\bigotimes_{i \in I} ! \left(\underbrace{!\,o \multimap \cdots \multimap !\,o}_{ar_i} \multimap o \right) \multimap o \right]\!\!\right]$$

where the return type o is naturally interpreted as the set of states $[\![o]\!] = Q$ of the alternating tree automaton. As explained in the introduction, the transition function δ of the alternating tree automaton $\mathcal{A}$ is itself interpreted as a subset

$$[\![\delta]\!] \quad \subseteq \quad \left[\!\!\left[\bigwith_{i \in I} \left(\underbrace{!\,o \multimap \cdots \multimap !\,o}_{ar_i} \multimap o \right) \right]\!\!\right]$$

which may be "strengthened" in the categorical sense as a subset

$$[\![\delta^\dagger]\!] \quad \subseteq \quad \left[\!\!\left[\bigotimes_{i\in I} \; ! \; \left(\underbrace{!\,o \multimap \cdots \multimap !\,o}_{ar_i} \multimap o \right) \right]\!\!\right]$$

where we turn to our advantage the well-known isomorphism of linear logic:

$$!\,(\,A \,\&\, B\,) \quad \cong \quad !A \otimes !B.$$

As explained in the introduction, a first contribution of the article is to establish the following result in the case of the traditional relational semantics of linear logic, extended here with a fixpoint operator Y:

▶ **Theorem 2.** *An alternating tree automaton $\mathcal{A}$ with a set of states Q has a finite accepting run-tree with initial state q_0 over the possibly infinite tree generated by a λY-term t if and only if there exists $u \in [\![\delta^\dagger]\!]$ such that $(u, q_0) \in [\![t]\!]$, where $[\![\delta^\dagger]\!] = \mathcal{M}_{fin}([\![\delta]\!])$ denotes the set of finite multisets of elements of $[\![\delta]\!]$.*

Another equivalent way to state the theorem is that the set of initial states from which there exists a finite run-tree of the alternating tree automaton $\mathcal{A}$ coincides with the result of composing $[\![t]\!]$ and $[\![\delta^\dagger]\!]$ in the relational semantics. At this point, it appears that the only hurdle towards an extension of this theorem to the alternating tree automata with coinductive (rather than inductive) acceptance condition is the *finiteness* of multiplicities in the traditional relational interpretation of the exponential modality. For this reason, the authors developed in a companion paper [10] an *infinitary* variant <u>*Rel*</u> of the relational model of linear logic, where the exponential modality noted there « $\natural$ » in order to distinguish it from the traditional « ! » transports every set A (of cardinality required to be smaller than the reals) to the set

$$\natural A \quad = \quad \mathcal{M}_{count}(A)$$

of *finite-or-countable* multisets of elements of A. In this alternative relational semantics, there is a *coinductive* fixpoint operator Y satisfying the equations of a Conway operator, and thus providing an interpretation of the λY-calculus. The infinitary interpretation of a λY-term is denoted $[\![t]\!]_\natural$ in order to distinguish it from the traditional finitary interpretation. Note that the interpretation $[\![\delta]\!]_\natural = [\![\delta]\!]$ is unchanged, and that its strengthening to $[\![\delta^\dagger]\!]_\natural$ reflects now the infinitary principles of the model. In particular, it is possible to detect whether a transition has been called a countable number of times. This brings us to the second main contribution of this article, which is to adapt the previous theorem for finite accepting run-trees to the general case of possibly infinite accepting run-trees:

▶ **Theorem 3.** *An alternating tree automaton $\mathcal{A}$ with a set of states Q has a possibly infinite accepting run-tree with initial state q_0 over the possibly infinite tree generated by a λY-term t if and only if there exists $u \in [\![\delta^\dagger]\!]_\natural$ such that $(u, q_0) \in [\![t]\!]_\natural$, where $[\![\delta^\dagger]\!]_\natural = \mathcal{M}_{count}([\![\delta]\!]_\natural)$ denotes the set of finite-or-countable multisets of elements of $[\![\delta]\!]_\natural$.*

This theorem should be understood as a purely semantic counterpart to Theorem 1. The connection is provided by the foundational and elegant work by Bucciarelli and Ehrhard [2, 3] on indexed linear logic, which establishes a nice correspondence between the elements of the relational semantics and a finitary variant of intersection types. By shifting from finite to finite-or-countable multisets and intersection types, we are able to recover here the discriminating power of general alternating tree automata. In particular, the set of

initial states of the alternating tree automaton $\mathcal{A}$ from which there exists a (finite or infinite) accepting run-tree is equal to the composition of $[\![t]\!]_\ell$ and of $[\![\delta^\dagger]\!]_\ell$ in our infinitary variant of the relational semantics.

We should mention however that there is a minor difference between our semantic result and the original Theorem 1, related to the fact that Kobayashi chose to work in a type system where intersection is understood as an idempotent operation. This choice is motivated in his work by the desire to keep the type system finitary, and thus to obtain decidability results. We prefer to work here with an infinitary relational semantics, corresponding to an infinitary and non-idempotent variant of Kobayashi's intersection type system. The reason is that shifting from an infinitary to an idempotent intersection type system corresponds from a semantic point of view to shifting from an infinitary relational semantics to its extensional collapse. Ehrhard [6] has recently established that the extensional collapse of the relational semantics is provided by a lattice model, where the formulas of linear logic are interpreted as partially ordered sets. This means that the corresponding intersection type systems should include a subtyping relation, as well-understood for instance by Terui in [27]. This subtyping relation is not mentioned in the original work by Kobayashi [16] nor in the later work by Kobayashi and Ong [17] and although their final result is certainly valid, this omission has lead to much confusion.

4 A type-theoretic account of alternating parity automata

4.1 Colored intersection types

After designing in [16] the type-theoretic approach to alternating tree automata recalled in the previous section, Kobayashi carried on in this direction and generalized it with Ong [17] to the larger class of alternating *parity* automata. The basic idea of this work is to incorporate coloring annotations in the intersection types, in order to reflect in the type system the parity conditions of the tree automata. Suppose for instance that a binary terminal $a \in \Sigma$ induces a transition $\delta(q,a) = (1,q_1) \wedge (2,q_2)$ in an alternating parity tree automaton with coloring function $\Omega : Q \to \mathbb{N}$. In that case, the terminal a is assigned in [17] the intersection type

$$a \quad : \quad (q_1,m_1) \to (q_2,m_2) \to q \tag{12}$$

where $m_1 = max(\Omega(q_1),\Omega(q))$ and $m_2 = max(\Omega(q_2),\Omega(q))$ are colors indicating to the type system the colors of the states q, q_1, q_2 of the parity tree automaton. In order to prepare the later development of paper, we find useful to simplify the colored intersection type system originally formulated by Kobayashi and Ong, and to stress at the same time the modal nature of colors (or priorities) in higher-order model-checking. So, given a set of states Q and a coloring function $\Omega : Q \to \mathbb{N}$, we define the set of colors

$$Col \quad = \quad \{\, \Omega(q) \mid q \in Q \,\} \ \uplus \ \{\, \epsilon \,\}$$

which contains the colors used by Ω, together with an additional color ϵ which will play the role of neutral element. The intersection types are then generated by the grammar

$$\begin{array}{lcll} \theta & ::= & q \ \mid \ \tau \to \theta & (q \in Q) \\ \tau & ::= & \bigwedge_{i\in I} \boxdot_{m_i} \theta_i & (I \text{ finite}, m_i \in Col) \end{array}$$

The refinement relation between intersection types and kinds is defined by the inductive rules below:

$$\frac{q \in Q}{q :: o} \qquad \frac{\tau :: \kappa_1 \qquad \theta :: \kappa_2}{\tau \to \theta :: \kappa_1 \to \kappa_2} \qquad \frac{\forall i \in I \quad \theta_i :: \kappa}{\bigwedge_{i\in I} \boxdot_{m_i} \theta_i :: \kappa}$$

$$\text{Axiom}\quad \frac{}{x \,:\, \bigwedge_{\{i\}} \boxdot_\epsilon\, \theta_i \,::\, \kappa \;\vdash\; x \,:\, \theta_i \,::\, \kappa} \qquad (x \in \mathcal{V} \cup \mathcal{N})$$

$$\delta\quad \frac{\{\,(i, q_{ij}) \mid 1 \le i \le n, 1 \le j \le k_i\,\} \quad \text{satisfies} \quad \delta_A(q,a)}{\emptyset \vdash a \,:\, \bigwedge_{j=1}^{k_1} \boxdot_{\Omega(q_{1j})}\, q_{1j} \;\to\; \ldots \;\to\; \bigwedge_{j=1}^{k_n} \boxdot_{\Omega(q_{nj})}\, q_{nj} \to q \,::\, o \to \cdots \to o \to o} \qquad a \in \Sigma$$

$$\text{App}\quad \frac{\Delta \vdash t : (\boxdot_{m_1}\, \theta_1 \;\wedge \cdots \wedge\, \boxdot_{m_k}\, \theta_k) \to \theta \,::\, \kappa \to \kappa' \qquad \Delta_1 \vdash u \,:\, \theta_1 \,::\, \kappa \cdots \Delta_k \vdash u \,:\, \theta_k \,::\, \kappa}{\Delta \,+\, \boxdot_{m_1}\Delta_1 \,+\, \ldots \,+\, \boxdot_{m_k}\Delta_k \;\vdash\; t\,u \,:\, \theta \,::\, \kappa'}$$

$$\lambda\quad \frac{\Delta\,,\, x \,:\, \bigwedge_{i\in I} \boxdot_{m_i}\, \theta_i \,::\, \kappa \;\vdash\; t \,:\, \theta \,::\, \kappa' \qquad I \subseteq J}{\Delta \;\vdash\; \lambda x\,.\,t \,:\, \left(\bigwedge_{j\in J} \boxdot_{m_j}\, \theta_j\right) \to \theta \,::\, \kappa \to \kappa'}$$

Figure 1 The type system $\gimel(\mathcal{A})$ associated to the alternating parity tree automaton $\mathcal{A}$.

Note that the color modality acts on intersection types and contexts by

$$\boxdot_m \left(\bigwedge_{i\in I} \boxdot_{m_i} \theta_i\right) \;=\; \bigwedge_{i\in I} \boxdot_{\max(m,m_i)} \theta_i \qquad\qquad \boxdot_m (\,x : \tau\,,\, \Delta\,) \;=\; x : \boxdot_m \tau\;,\; \boxdot_m \Delta$$

Note also that the neutral color ϵ is only introduced here to allow a uniform definition of types and contexts. It does not affect the coloring of types, and should be understood as the *absence* of a coloring annotation. From this, one obtains an intersection type system $\gimel(\mathcal{A})$ parametrized by the alternating tree automaton $\mathcal{A}$, whose rules are given in Figure 1. Here we use the Hebrew letter $\gimel$ which should be read "tsadi". The resulting type system $\gimel(\mathcal{A})$ enables us to type the rewriting rules of a higher-order recursion scheme

$$\Delta \;\vdash\; \mathcal{R}(F) \,:\, \sigma \,::\, \kappa \tag{13}$$

where the non-terminals occurring in the λ-term $\mathcal{R}(F)$ appear as variables in the context Δ of the typing judgement. On the other hand, the intersection type system $\gimel(\mathcal{A})$ does not include a fixpoint operator Y and for that reason does not accomodate recursion.

4.2 Interpretation of recursion

In order to accomodate recursion in the intersection type system $\gimel(\mathcal{A})$, we need to extend it with a rule fix whose purpose is to expand the non-terminals $F \in \mathcal{N}$ of the recursion scheme $\mathcal{G}$ in order to obtain possibly infinitary derivation trees. So, given a higher-order recursion scheme $\mathcal{G}$ and an alternating parity automaton $\mathcal{A}$, we define the intersection type system $\gimel_{fix}(\mathcal{G}, \mathcal{A})$ as $\gimel(\mathcal{A})$ where we add the recursion rule

$$fix\qquad \frac{\Delta \vdash \mathcal{R}(F) \,:\, \theta \,::\, \kappa}{F \,:\, \boxdot_\epsilon\, \theta \,::\, \kappa \;\vdash\; F \,:\, \theta \,::\, \kappa} \qquad \mathrm{dom}(\Gamma) \subseteq \mathcal{N}$$

and at the same time restrict the Axiom rule to variables $x \in \mathcal{V}$, and in particular do not allow the Axiom rule to be applied on non-terminals any more. Note that the context Δ does not appear in the conclusion of the rule fix, but only in its premise. The fix rule is thus akin to an axiom rule adapted to handle the non-terminals of the higher-order recursion scheme.

An important aspect of the resulting intersection type system $\gimel_{fix}(\mathcal{G}, \mathcal{A})$ is that its derivation trees may be of countable depth. As in Kobayashi's original type system, this infinitary nature of the intersection type system enables one to reflect the existence of infinitary runs

in the alternating parity automaton $\mathcal{A}$. In order to articulate the parity condition of the automaton with the typing derivations, a color is assigned to each node of the derivation tree, in the following way:

- the node $\Delta_i \vdash u : \theta_i :: \kappa$ is assigned the color m_i in every Application rule

$$\frac{\Delta \vdash t : (\boxdot_{m_1} \theta_1 \wedge \cdots \wedge \boxdot_{m_k} \theta_k) \to \theta :: \kappa \to \kappa' \qquad \cdots \quad \Delta_i \vdash u : \theta_i :: \kappa \quad \cdots}{\Delta + \boxdot_{m_1}\Delta_1 + \ldots + \boxdot_{m_k}\Delta_k \vdash t\,u : \theta :: \kappa'}$$

  of the derivation tree,
- all the other nodes of the derivation tree are assigned the neutral color ϵ, which means in some sense that they are not colored by the typing system.

A nice aspect of our approach compared to the original formulation in [17] is that the parity condition traditionally applied to the alternating parity automaton $\mathcal{A}$ extends to a very simple parity condition on the derivation trees of $\boldsymbol{\gamma}_{fix}(\mathcal{G}, \mathcal{A})$. Indeed, the color of an infinite branch of a given derivation tree can be defined as

- the neutral color ϵ if no other color $m \in Col$ occurs infinitely often in the branch,
- otherwise, the maximal non-neutral color $m \in Col \setminus \{\epsilon\}$ seen infinitely often.

Then, an infinite branch of the derivation tree is declared *winning* precisely when its color is an even integer (and in particular different from the neutral color). A winning derivation tree is then defined as a derivation tree whose infinite branches are all winning in the sense just explained.

4.3 Soundness and completeness

Once the notion of infinite winning derivation tree explicated, as we have just done in the previous section, there remains to relate this winning condition to the acceptance condition of alternating parity automata. To that purpose, and for the sake of the presentation, we choose to restrict ourself to *productive* recursion schemes, as it is also done in [17]. Note that it is only a very mild restriction, since every recursion scheme $\mathcal{G}$ can be transformed in to a productive recursion scheme $\mathcal{G}'$ which outputs a special leaf symbol Ω whenever the Böhm evaluation of the original scheme $\mathcal{G}$ would have infinitely looped. The following theorem establishes a soundness and completeness theorem which relates the winning condition on the infinite derivation trees of $\boldsymbol{\gamma}_{fix}(\mathcal{G}, \mathcal{A})$ to the parity acceptance condition of the automaton $\mathcal{A}$ during its exploration of the infinite tree $[\![\mathcal{G}]\!]$ generated by the recursion scheme $\mathcal{G}$:

▶ **Theorem 4** (soundness and completeness)**.** *Suppose given a productive recursion scheme $\mathcal{G}$ and an alternating parity automaton $\mathcal{A}$. There exists a winning run-tree of $\mathcal{A}$ over $[\![\mathcal{G}]\!]$ with initial state q if and only if the sequent*

$$S : \boxdot_\epsilon q :: o \vdash S : q :: o \tag{14}$$

has a winning derivation tree in the type system $\boldsymbol{\gamma}_{fix}(\mathcal{G}, \mathcal{A})$.

There are several ways to establish the theorem. One possible way is to establish an equivalence with the original soundness and completeness by Kobayashi and Ong [17]. One should be careful however that the original proof in [17] was incomplete, and has been corrected in the (unpublished) journal version of the paper. In order to establish the equivalence, one shows that the existence of a winning derivation tree of the sequent (14) in the intersection type system $\boldsymbol{\gamma}_{fix}(\mathcal{G}, \mathcal{A})$ is equivalent to the existence of a winning strategy for Eve in the parity game defined in [17]. Another more direct proof is possible, based on the reformulation by Haddad [12] of Kobayashi and Ong's treatment of infinitary (rather than

simply finitary) sequences of rewrites on higher-order recursion schemes. It is in particular important to observe that the infinitary nature of computations requires to extend the usual soundness and completeness arguments based on Böhm trees, finite rewriting sequences and continuity. This point was apparently forgotten in [17] and corrected in the journal version of the paper.

5 An indexed tensorial logic with colors

The notation $\boxdot_m \theta$ is used in our intersection type system $\gamma_{fix}(\mathcal{G}, \mathcal{A})$ as a way to stress the modal nature of colors, and it replaces for the better the notation (θ, m) used by Kobayashi and Ong in [17]. As we will see, the discovery of the modal nature of colors is fundamental, and is not just a matter of using the appropriate notation. In particular, it enables us to simplify both technically and conceptually the original intersection type system in [17]. By way of illustration, the original intersection type (12) of the binary terminal $a \in \Sigma$ considered in §4.1 is replaced by the simpler intersection type:

$$a \quad : \quad \boxdot_{n_1}\, q_1 \to \boxdot_{n_2}\, q_2 \to\, q \tag{15}$$

where $n_1 = \Omega(q_1)$ and $n_2 = \Omega(q_2)$. Interestingly, the color of the state q is not mentioned in the type anymore. The reason is that this alternative account of colors achieved in our type system is not just "simpler" than the original one: it also reveals a deep and somewhat unexpected connection with linear logic, since as we will see, this "disappearance" of the color $\Omega(q)$ in (15) is related to the well-known linear decomposition $A \Rightarrow B = !A \multimap B$ of the intuitionistic implication in linear logic. One essential difference however is that the exponential modality « ! » of linear logic is replaced by a family of modal boxes $\Omega(m)$ which formally defines what Melliès calls a *parametric comonad* in [21][19].

This key observation enables us to translate the intersection type system $\gamma_{fix}(\mathcal{G}, \mathcal{A})$ into an infinitary variant of linear logic equipped with a family of color modalities noted $\Box_m$ for $m \in \mathbb{N}$. A nice feature of the translation is that it transports the intersection type system $\gamma_{fix}(\mathcal{G}, \mathcal{A})$ which depends on $\mathcal{G}$ and $\mathcal{A}$ into an intersection type system which does not depend on them anymore — although it still depends on the set Q of states of the automaton. The infinitary variant of linear logic which we use for the translation is

- *indexed* in the sense of Bucciarelli and Ehrhard [2, 3]. In particular, the finite or countable intersection types $\bigwedge_{i\in I} \theta_i$ of $\gamma_{fix}(\mathcal{G}, \mathcal{A})$ are translated as finite or countable indexed families $[\theta_i \mid i \in I]$ of formulas of the logic,
- *tensorial* in the sense of Melliès [22, 20, 21]. In this specific case, every negated formula of the logic is negated with respect to a specific state $q \in Q$ of the automaton, and is thus of the form $\sigma \multimap q$, which may be alternatively written as $\neg_q \sigma$ or even as $\overset{q}{\neg}\, \sigma$.

In this way, one obtains an indexed and colored variant of tensorial logic, called $\mathrm{LT}(Q)$ in the sequel, and whose formulas are inductively generated by the following grammar:

$$A, B \quad ::= \quad 1 \quad | \quad A \otimes B \quad | \quad \neg_q A \quad | \quad \Box_m A \quad | \quad [\, A_j \mid j \in J\,] \qquad (m \in Col,\ q \in Q)$$

As already mentioned, following the philosophy in [3], the finite or countable indexed set $[\sigma_j \mid j \in J]$ internalizes the intersection operator of $\gamma_{fix}(\mathcal{G}, \mathcal{A})$ in our indexed tensorial logic, see our companion paper [9] for details. Importantly, the resulting indexed logic $TL(Q)$ can be used as an intersection type system refining the simply-typed λ-calculus in just the same way as $\gamma_{fix}(\mathcal{G}, \mathcal{A})$, see the Appendix . In particular, the fact that $\Box$ defines a parametric monoidal comonad in the logic means that the sequents

$$\begin{array}{ccc} \Box_\epsilon\ A & \vdash & A \\ \Box_{\max(m_1,\,m_2)}\ A & \vdash & \Box_{m_1}\ \Box_{m_2}\ A \\ \Box_m\ A\ \otimes\ \Box_m\ B & \vdash & \Box_m\ (A\ \otimes\ B) \end{array}$$

are provable for all colors $m, m_1, m_2 \in \mathbb{N}$, and all formulas A, B. In order to deal with recursion schemes, we admit derivation trees with finite or countable depth in the logical system $TL(Q)$. The nodes of the derivation trees of $TL(Q)$ are then colored in the following way:

- every node $\Gamma \vdash M\ :\ A\ ::\ \kappa$ in a Right introduction of the modality $\Box_m$:

$$\frac{\Gamma \vdash\ M\ :\ A\ ::\ \kappa}{\Box_m \Gamma \vdash\ M\ :\ \Box_m\, A\ ::\ \kappa}\quad \text{Right } \Box_m$$

  is assigned the color m of the modality,
- all the other nodes of the derivation tree are assigned the neutral color ϵ.

The winning condition on an infinite derivation tree of $TL(Q)$ is then directly adapted from the similar condition in $\gamma_{fix}(\mathcal{G}, \mathcal{A})$. Thanks to this condition, we are ready to state a useful correspondence theorem between $\gamma_{fix}(\mathcal{G}, \mathcal{A})$ and $TL(Q)$ for any (productive) recursion scheme $\mathcal{G}$. Suppose that for each $F \in \mathcal{N}$ of kind $\kappa(F)$ of the recursion scheme $\mathcal{G}$, we introduce a new free variable $freeze(F)$ of kind $\kappa(F) \to \kappa(F)$; that we replace each λ-term $\mathcal{R}(F)$ by its $\beta\eta$-long normal form ; and finally, that we substitute each occurrence of F appearing in any $\beta\eta$-long normal form $\mathcal{R}(G)$ of the recursion scheme $\mathcal{G}$ with the λ-term $freeze(F)\ F$ of the same kind $\kappa(F)$. This transformation induces a context-free grammar of « blocks» consisting of the $\beta\eta$-long $\mathcal{R}(G)$'s, which generates an infinite λ-term in $\beta\eta$-long normal form, noted $term(\mathcal{G})$, with free variables of the form $freeze(F)$. Moreover, this infinite λ-term $term(\mathcal{G})$ is coinductively typed in the simply-typed λ-calculus by the typing judgment:

$$\ldots\quad ,\quad freeze(F) : \kappa(F) \to \kappa(F)\quad ,\quad \ldots\quad \vdash\quad term(\mathcal{G})\quad :\quad o \tag{16}$$

where F runs over all the non-terminals $F \in \mathcal{N}$ of the higher-order recursion scheme $\mathcal{G}$. At this point, we are ready to recast our Theorem 4 in the proof-theoretic language of indexed tensorial logic:

▶ **Theorem 5.** *There exists a winning derivation tree in $\gamma_{fix}(\mathcal{G}, \mathcal{A})$ of the sequent*

$$S\ :\ \boxdot_\epsilon\ q_0\ ::\ o\ \vdash\ S\ :\ q_0\ ::\ o \tag{17}$$

if and only if there exists a winning derivation tree in $TL(Q)$ of a sequent

$$\Gamma\ \vdash\quad term(\mathcal{G})\ :\ q_0\ ::\ o \tag{18}$$

refining the typing judgment (16).

6 Putting all together: relational semantics of linear logic and higher-model model-checking

Once the connection between higher-order model-checking and indexed tensorial logic established in §5, there remains to exhibit the associated relational semantics of linear logic, following the ideas of Bucciarelli and Ehrhard [2, 3]. This trail leads us to an infinitary and colored variant of the usual relational semantics of linear logic, developed in our companion paper [10]. The key observation guiding the construction is that the functor

$$\Box\quad :\quad A\quad \mapsto\quad Col \times A\quad :\quad Rel\quad \to\quad Rel$$

equipped with the coercion maps

$$\begin{array}{rcrcl}
\{(((m,a),(m,b)),(m,(a,b))) \mid a\in A,\, b\in B,\, m\in Col\} & : & \Box A\otimes \Box B & \to & \Box(A\otimes B)\\
\{(\star,(m,\star)) \mid m\in Col\} & : & 1 & \to & \Box 1\\
\{((\max(m_1,m_2),a),(m_1,(m_2,a))) \mid a\in A\} & : & \Box A & \to & \Box\Box A\\
\{((\epsilon,a),a) \mid a\in A\} & : & \Box A & \to & A
\end{array}$$

defines a lax monoidal comonad $\Box : \mathit{Rel} \to \mathit{Rel}$ on the category Rel of sets and relations. Moreover, the comonad distributes (or better: commutes) with the exponential modality $!$, in such a way that these two comonads compose into a new exponential modality of linear logic $\mathbf{!}$ defined by the equation $\mathbf{!}A = !\Box A$. A Conway operator Y can be then defined in order to reflect in the relational semantics the definition of the winning condition on the infinite derivations of $TL(Q)$. This fixpoint operator can be seen as a combination of the inductive and coinductive fixpoints of the model, where the color of an input indicates whether the fixpoint operator should be defined inductively (when the color is odd or neutral) or coinductively (when the color is even). The relational interpretation of a λY-term t in this infinitary model is denoted as $[\![t]\!]_{\mathbf{!}}$. The interpretation $[\![\delta]\!]_{\mathbf{!}}$ of the transition function δ is defined similarly as for $[\![\delta]\!]_{!}$, except that the color information is incorporated in the semantics following the comonadic principles underlying the translation (15) in §5. Typically, the transition (11) is interpreted in the colored relational semantics as

$$([\,],([(\Omega(q_0),q_0),(\Omega(q_1),q_1)],q_0)) \quad\in\quad [\![\delta]\!]_{\mathbf{!}}$$

The last contribution of this paper, which underlies our companion paper [10] but is not stated there, establishes a clean correspondence between the relational semantics of a higher-order recursion scheme $\mathcal{G}$ (seen below as a λY-term $t_{\mathcal{G}}$) and the exploration of the associated ranked tree $[\![\mathcal{G}]\!]$ by an alternating *parity* automaton $\mathcal{A}$:

▶ **Theorem 6.** *An alternating* parity *tree automaton $\mathcal{A}$ with a set of states Q has a* winning *run-tree with initial state q_0 over the ranked tree $[\![\mathcal{G}]\!]$ generated by the λY-term $t_{\mathcal{G}}$ if and only if there exists $u \in [\![\delta^\dagger]\!]_{\mathbf{!}}$ such that $(u,q_0) \in [\![t_{\mathcal{G}}]\!]_{\mathbf{!}}$, where $[\![\delta^\dagger]\!]_{\mathbf{!}} = \mathcal{M}_{count}(Col \times [\![\delta]\!]_{\mathbf{!}})$ denotes the set of finite-or-countable colored multisets of elements of $[\![\delta]\!]_{\mathbf{!}}$.*

7 Related works

The field of higher-order model-checking was to a large extent started at the turn of the century by Knapik, Niwinski, Urzyczyn, who established that for every $n \geq 0$, Σ-labelled trees generated by order-n safe recursion schemes are exactly those that are generated by order-n pushdown automata, and further, that they have decidable MSO theories. The safety condition was relaxed a few years later by Ong, who established the MSO decidability for general order-n recursion schemes, using ideas imported from game semantics. Unfortunately, Ong's proof was intricate and somewhat difficult to understand. Much work was thus devoted to establish the decidability result by other means. Besides the type-theoretic approach initiated by Kobayashi [16, 17], Hague, Murawski, Ong, Serre [13] developed an automata-theoretic approach based on the translation of the higher-order recursion scheme $\mathcal{G}$ into a collapsible pushdown automaton (CPDA), which led the four authors to another proof of MSO decidability for order-n recursion schemes. A clarifying connection was then made by Salvati and Walukiewicz between this translation of higher-order recursion schemes into CPDAs and the traditional evaluation mechanism of the environment Krivine machine [24]. Following this discovery, Salvati and Walukiewicz are currently developing a semantic approach to higher-order model checking, based on the interpretation in finite models of the λ-calculus with fixpoint operators, see [26, 25] for details. The idea of connecting linear logic to automata theory is a longstanding dream which has been nurtured by a number of important

contributions. Among them, we would like to mention the clever work by Terui [27] who developed a semantic and type-theoretic approach based on linear logic, intersection types and automata theory in order to characterize the complexity of evaluation in the simply-typed λ-calculus. In a different but related line of work, explicitly inspired by Bucciarelli and Ehrhard's indexed linear logic [2, 3], de Carvalho [5] establishes an interesting correspondence between intersection types and the length of evaluation in a Krivine machine.

8 Conclusions and perspectives

The purpose of the present paper is to connect higher-order model-checking to a series of advanced ideas in contemporary semantics, like linear logic and its relational semantics, indexed linear logic, distributive laws and parametric comonads. All these ingredients meet and combine surprisingly well. The approach reveals in particular that the traditional treatment of inductive-coinductive reasoning based on colors (or priorities) is secretly based on the same comonadic principles as the exponential modality of linear logic.

Besides the conceptual promises offered by these connections, we would like to conclude the paper by mentioning that this stream of ideas leads us to an alternative and purely semantic proof of MSO decidability for higher-order recursion schemes, after [23, 17, 13]. The basic idea is to replace the infinitary colored relational semantics constructed in §6 by a finitary variant based on the prime-algebraic lattice semantics of linear logic. From a type-theoretic point of view, this lattice semantics corresponds to an intersection type system with subtyping for linear logic recently formulated by Terui [27]. We have shown in a companion paper [11] how to recover the MSO decidability result for order-n recursion schemes by adapting to this finitary semantics of linear logic the constructions performed here for its relational semantics. One interesting feature of the resulting model of the λY-calculus is that a morphism $D \to E$ in the Kleisli category consists in a continuous function

$$f \quad : \quad \underbrace{D \times \cdots \times D}_{n} \quad \longrightarrow \quad E$$

where n is the number of colors considered in the semantics ; and that the fixpoint Yf of a morphism $f : D \to D$ is defined in that case by the alternating formula

$$Yf \quad = \quad \nu x_n \,.\, \mu x_{n-1} \,\ldots\, \nu x_2 \,.\, \mu x_1 \,.\, \nu x_0 . f(x_0, \ldots, x_n)$$

where we suppose (without loss of generality) that n is even, where μ and ν denote the least and greatest fixpoint operators, respectively. We believe that the apparition of this simple formula and the fact that it defines a Conway operator Y and thus a model of the λY-calculus is a key contribution to the construction of a semantic and purely compositional account of higher-order model-checking.

References

1 Klaus Aehlig. A finite semantics of simply-typed lambda terms for infinite runs of automata. *Logical Methods in Computer Science*, 3(3), 2007.

2 Antonio Bucciarelli and Thomas Ehrhard. On phase semantics and denotational semantics in multiplicative-additive linear logic. *Ann. Pure Appl. Logic*, 102(3):247–282, 2000.

3 Antonio Bucciarelli and Thomas Ehrhard. On phase semantics and denotational semantics: the exponentials. *Ann. Pure Appl. Logic*, 109(3):205–241, 2001.

4 M. Coppo, M. Dezani-Ciancaglini, F. Honsell, and G. Longo. Extended Type Structures and Filter Lambda Models. In *Logic Colloquium 82*, 1984.

5 Daniel de Carvalho. Execution time of lambda-terms via denotational semantics and intersection types. *CoRR*, abs/0905.4251, 2009.

6 Thomas Ehrhard. The scott model of linear logic is the extensional collapse of its relational model. *Theor. Comput. Sci.*, 424:20–45, 2012.

7 E. Allen Emerson and Charanjit S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991*, pages 368–377. IEEE Computer Society, 1991.

8 Jean-Yves Girard, Paul Taylor, and Yves Lafont. *Proofs and Types*. C.U.P., 1989.

9 Charles Grellois and Paul-André Melliès. Indexed linear logic and higher-order model checking. In *Intersection Types and Related Systems 2014*, EPTCS 177, 2015.

10 Charles Grellois and Paul-André Melliès. An infinitary model of linear logic. In Andrew M. Pitts, editor, *FoSSaCS*, volume 9034 of *LNCS*, 2015.

11 Charles Grellois and Paul-André Melliès. Finitary semantics of linear logic and higher-order model-checking, MFCS 2015.

12 Axel Haddad. *Shape-preserving transformations of higher-order recursion schemes*. PhD thesis, Université Paris Diderot, 2013.

13 Matthew Hague, Andrzej S. Murawski, C.-H. Luke Ong, and Olivier Serre. Collapsible pushdown automata and recursion schemes. In *LICS*, 2008.

14 Haruo Hosoya, Jerome Vouillon, and Benjamin C. Pierce. Regular expression types for XML. In Martin Odersky and Philip Wadler, editors, *ICFP*, 2000.

15 David Janin and Igor Walukiewicz. On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic. In Ugo Montanari and Vladimiro Sassone, editors, *CONCUR'96 Proceedings*, volume 1119 of *LNCS*, pages 263–277. Springer, 1996.

16 Naoki Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In Zhong Shao and Benjamin C. Pierce, editors, *POPL*, 2009.

17 Naoki Kobayashi and C.-H. Luke Ong. A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. In *LICS*, 2009.

18 Paul-André Melliès. Parametric monads and enriched adjunctions. LOLA 2012.

19 Paul-André Melliès. Functorial boxes in string diagrams. In Zoltán Ésik, editor, *CSL*, volume 4207 of *Lecture Notes in Computer Science*, pages 1–30. Springer, 2006.

20 Paul-André Melliès. Game semantics in string diagrams. In *LICS*, 2012.

21 Paul-André Melliès. The parametric continuation monad. *Math. Struct. Comp. Sci.*, 2014.

22 Paul-André Melliès and Nicolas Tabareau. Resource modalities in game semantics. In *LICS*, pages 389–398. IEEE Computer Society, 2007.

23 C.-H. Luke Ong. On model-checking trees generated by higher-order recursion schemes. In *LICS*, pages 81–90. IEEE Computer Society, 2006.

24 Sylvain Salvati and Igor Walukiewicz. Recursive schemes, krivine machines, and collapsible pushdown automata. In *RP*, 2012.

25 Sylvain Salvati and Igor Walukiewicz. Evaluation is msol-compatible. In *FSTTCS*, volume 24 of *LIPIcs*, 2013.

26 Sylvain Salvati and Igor Walukiewicz. Using models to model-check recursive schemes. In *TLCA*, volume 7941 of *LNCS*, 2013.

27 Kazushige Terui. Semantic evaluation, intersection types and complexity of simply typed lambda calculus. In *RTA*, volume 15 of *LIPIcs*, 2012.

A Van Benthem Theorem for Modal Team Semantics*

Juha Kontinen[1], Julian-Steffen Müller[2], Henning Schnoor[3], and Heribert Vollmer[2]

1 **University of Helsinki, Department of Mathematics and Statistics, P.O. Box 68, 00014 Helsinki, Finland**
2 **Leibniz Universität Hannover, Institut für Theoretische Informatik, Appelstr. 4, 30167 Hannover, Germany**
3 **Institut für Informatik, Christian-Albrechts-Universität zu Kiel, 24098 Kiel, Germany**

Abstract

The famous van Benthem theorem states that modal logic corresponds exactly to the fragment of first-order logic that is invariant under bisimulation. In this article we prove an exact analogue of this theorem in the framework of modal dependence logic MDL and team semantics. We show that modal team logic MTL, extending MDL by classical negation, captures exactly the FO-definable bisimulation invariant properties of Kripke structures and teams. We also compare the expressive power of MTL to most of the variants and extensions of MDL recently studied in the area.

1998 ACM Subject Classification F.4.1 [Mathematical Logic] Modal Logic

Keywords and phrases modal logic, dependence logic, team semantics, expressivity, bisimulation, independence, inclusion, generalized dependence atom

Digital Object Identifier 10.4230/LIPIcs.CSL.2015.277

1 Introduction

The concepts of dependence and independence are ubiquitous in many scientific disciplines such as experimental physics, social choice theory, computer science, and cryptography. Dependence logic D [21] and its so-called team semantics have given rise to a new logical framework in which various notions of dependence and independence can be formalized and studied. Dependence logic extends first-order logic by dependence atoms

$$=(x_1, \ldots, x_{n-1}, x_n)\,, \tag{1}$$

expressing that the value of the variable x_n is functionally dependent on the values of $x_1, \ldots, x_{n-1}$. The formulas of dependence logic are evaluated over *teams*, i. e., sets of assignments, and not over single assignments as in first-order logic.

In [22] a modal variant of dependence logic MDL was introduced. In the modal framework teams are sets of worlds, and a dependence atom

$$=(p_1, \ldots, p_{n-1}, p_n) \tag{2}$$

* The first author was supported by the Academy of Finland grants 264917 and 275241.

holds in a team T if there is a Boolean function that determines the value of the propositional variable p_n from those of $p_1, \ldots, p_{n-1}$ in all worlds in T. One of the fundamental properties of MDL (and of dependence logic) is that its formulas satisfy the so-called downwards closure property: if $M, T \models \varphi$, and $T' \subseteq T$, then $M, T' \models \varphi$. Still, the modal framework is very different from the first-order one, e.g., dependence atoms between propositional variables can be eliminated with the help of the classical disjunction $\veebar$ [22]. On the other hand, it was recently shown that eliminating dependence atoms using disjunction causes an exponential blow-up in the formula size, that is, any formula of $\mathsf{ML}(\veebar)$ logically equivalent to the atom in (2) is bound to have length exponential in n [10]. The central complexity theoretic questions regarding MDL have been solved in [20, 14, 3, 15].

Extended modal dependence logic, EMDL, was introduced in [4]. This extension is defined simply by allowing ML formulas to appear inside dependence atoms, instead of only propositions. EMDL can be seen as the first step towards combining dependencies with temporal reasoning. EMDL is strictly more expressive than MDL but its formulas still have the downwards closure property. In fact, EMDL has recently been shown to be equivalent to the logic $\mathsf{ML}(\veebar)$ [4, 10].

In the first-order case, several interesting variants of the dependence atoms have been introduced and studied. The focus has been on independence atoms

$$(x_1, \ldots, x_\ell) \perp_{(y_1, \ldots, y_m)} (z_1, \ldots, z_n),$$

and inclusion atoms

$$(x_1, \ldots, x_\ell) \subseteq (y_1, \ldots, y_\ell),$$

which were introduced in [9] and [5], respectively. The intuitive meaning of the independence atom is that the variables $x_1, \ldots, x_\ell$ and $z_1, \ldots, z_n$ are independent of each other for any fixed value of $y_1, \ldots, y_m$, whereas the inclusion atom declares that all values of the tuple $(x_1, \ldots, x_\ell)$ appear also as values of $(y_1, \ldots, y_\ell)$. In [11] a modal variant, MIL, of independence logic was introduced. The logic MIL contains MDL as a proper sublogic, in particular, its formulas do not in general have the downwards closure property. In [11] it was also noted that all MIL formulas are invariant under bisimulation when this notion is lifted from single worlds to a relation between sets of words in a natural way. At the same time (independently) in [10] it was shown that EMDL and $\mathsf{ML}(\veebar)$ can express exactly those properties of Kripke structures and teams that are downwards closed and invariant under k-bisimulation for some $k \in \mathbb{N}$.

A famous theorem by Johan van Benthem [23, 24] states that modal logic is exactly the fragment of first-order logic that is invariant under (full) bisimulation. In this paper we study the analogues of this theorem in the context of team semantics. Our main result shows that an analogue of the van Benthem theorem for team semantics can be obtained by replacing ML by *Modal Team Logic* (MTL). MTL was introduced in [16] and extends ML (and MDL) by classical negation $\sim$. More precisely, we show that for any team property P the following are equivalent:

- **(i)** There is an MTL-formula which expresses P,
- **(ii)** there is a first-order formula which expresses P and P is bisimulation-invariant,
- **(iii)** P is invariant under k-bisimulation for some k,
- **(iv)** P is bisimulation-invariant and local.

We also study whether all bisimulation invariant properties can be captured by natural variants of EMDL. We consider extended modal independence and extended modal inclusion logic (EMIL and EMINCL, respectively), which are obtained from EMDL by replacing the dependence atom with the independence (resp. inclusion) atom. We show that both of these logics fail to capture all bisimulation invariant properties, and therefore in particular

are strictly weaker than MTL. On the other hand, we show that EMINCL($\vee$) (EMINCL extended with classical disjunction) is in fact as expressive as MTL, but the analogously defined EMIL($\vee$) is strictly weaker. Finally, we show that the extension $\mathsf{ML}^{\mathsf{FO}}$ of ML by all first-order definable generalized dependence atoms (see [11]) gives rise to a logic that is as well equivalent to MTL. The full version of this paper (including all proofs) can be found in [12].

2 Preliminaries

A *Kripke model* is a tuple $M = (W, R, \pi)$ where W is a nonempty set of worlds, $R \subseteq W \times W$, and $\pi\colon P \to 2^W$, where P is the set of propositional variables. A *team* of a model M as above is simply a set $T \subseteq W$. The central basic concept underlying Väänänen's modal dependence logic and all its variants is that modal formulas are evaluated not in a world but in a team. This is made precise in the following definitions. We first recall the usual syntax of modal logic ML:

$$\varphi ::= p \mid \neg p \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi) \mid \Diamond\varphi \mid \Box\varphi,$$

where p is a propositional variable. Note that we consider only formulas in negation normal form, i.e., negation appears only in front of atoms. As will become clear from the definition of team semantics of ML, that we present next, p and $\neg p$ are not dual formulas, consequently *tertium non datur* does not hold in the sense that it is possible that $M, T \not\models p$ and $M, T \not\models \neg p$ (however, we still have that $M, T \models p \vee \neg p$ for all models M and teams T). It is worth noting that in [22], the connective $\neg$ is allowed to appear freely in MDL formulas (with semantics generalizing the atomic negation case of Definition 2.1 below, note that classical negation as allowed in MTL is not allowed in MDL). The well-known dualities from classical modal logic are also true for MDL formulas hence any ML-formula (even MDL) can be rewritten in such a way that $\neg$ only appears in front of propositional variables.

▶ **Definition 2.1.** Let $M = (W, R, \pi)$ be a Kripke model, let $T \subseteq W$ be a team, and let φ be an ML-formula. We define when $M, T \models \varphi$ holds inductively:

- If $\varphi = p$, then $M, T \models \varphi$ iff $T \subseteq \pi(p)$,
- If $\varphi = \neg p$, then $M, T \models \varphi$ iff $T \cap \pi(p) = \emptyset$,
- If $\varphi = \psi \vee \chi$ for some formulas ψ and χ, then $M, T \models \varphi$ iff $T = T_1 \cup T_2$ with $M, T_1 \models \psi$ and $M, T_2 \models \chi$,
- If $\varphi = \psi \wedge \chi$ for some formulas ψ and χ, then $M, T \models \varphi$ iff $M, T \models \psi$ and $M, T \models \chi$,
- If $\varphi = \Diamond\psi$ for some formula ψ, then $M, T \models \varphi$ iff there is some team T' of M such that $M, T' \models \psi$ and
  - for each $w \in T$, there is some $w' \in T'$ with $(w, w') \in R$, and
  - for each $w' \in T'$, there is some $w \in T$ with $(w, w') \in R$.
- If $\varphi = \Box\psi$ for some formula ψ, then $M, T \models \varphi$ iff $M, T' \models \psi$, where T' is the set $\{w' \in M \mid (w, w') \in R \text{ for some } w \in T\}$.

Analogously to the first-order setting, ML-formulas satisfy the following *flatness* property [21]. Here, the notation $M, w \models \varphi$ in item 3 refers to the standard semantics of modal logic (without teams).

▶ **Proposition 2.2.** *Let M be a Kripke model and T a team of M. Let φ be an* ML*-formula. Then the following are equivalent:*

1. $M, T \models \varphi$,
2. $M, \{w\} \models \varphi$ *for each* $w \in T$,
3. $M, w \models \varphi$ *for each* $w \in T$.

Modal team logic extends ML by a second type of negation, denoted by $\sim$, and interpreted just as classical negation. The syntax is formally given as follows:

$$\varphi ::= p \mid \neg p \mid \sim\varphi \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi) \mid \Diamond\varphi \mid \Box\varphi,$$

where p is a propositional variable. The semantics of MTL is defined by extending Def. 2.1 by the following clause:

- If $\varphi = \sim\psi$ for some formula ψ, then $M, T \models \varphi$ iff $M, T \not\models \psi$.

We note that usually (see [16]), MTL also contains dependence atoms; however since these atoms can be expressed in MTL we omit them in the syntax (see Proposition 2.3 below). The classical disjunction $\otimes$ (in some other context also referred to as *intuitionistic disjunction*) is also readily expressed in MTL: $\varphi \otimes \psi$ is logically equivalent to $\sim(\sim\varphi \wedge \sim\psi)$.

For an ML formula φ, we let φ^{dual} denote the formula that is obtained by transforming $\neg\varphi$ to negation normal form. Now by Proposition 2.2 it follows that

$$M, T \models \varphi^{dual} \text{ iff } M, w \not\models \varphi \text{ for all } w \in T,$$

hence $M, T \models \sim\psi^{dual}$ if and only if there is some $w \in T$ with $M, w \models \psi$. We therefore often write $\mathsf{E}\psi$ instead of $\sim\psi^{dual}$. Note that E is not a global operator stating existence of a world anywhere in the model, but E is evaluated in the current team. It is easy to see (and follows from Proposition 2.8) that a global "exists" operator cannot be expressed in MTL.

The next proposition shows that dependence atoms can be easily expressed in MTL.

▶ **Proposition 2.3.** *The dependence atom* (2) *can be expressed in* MTL *by a formula that has length polynomial in* n.

Proof. Note first that, analogously to the first-order case [1], (2) is logically equivalent with

$$\Big(\bigwedge_{1\le i\le n-1} {=}(p_i)\Big) \to {=}(p_n)\,,$$

where $\to$ is the so-called intuitionistic implication with the following semantics:

$$M, T \models \varphi \to \psi \text{ iff for all } T' \subseteq T\text{: if } M, T' \models \varphi \text{ then } M, T' \models \psi.$$

The connective $\to$ has a short logically equivalent definition in MTL (see [16]): $\varphi \to \psi$ is equivalent to $(\sim\varphi \otimes \psi) \otimes \bot$, where $\otimes$ is the dual of $\vee$, i.e., $\varphi \otimes \psi := \sim(\sim\varphi \vee \sim\psi)$, and $\bot$ is a shorthand for the formula $p_0 \wedge \neg p_0$. Finally, ${=}(p_i)$ can be written as $p_i \otimes \neg p_i$, hence the claim follows. ◀

The intuitionistic implication used in the proof above has been studied in the modal team semantics context in [25].

We now introduce the central concept of bisimulation [18, 24]. Intuitively, two *pointed models* (i.e., pairs of models and worlds from the model) (M_1, w_1) and (M_2, w_2) are bisimilar, if they are indistinguishable from the point of view of modal logic. The notion of k-bisimilarity introduced below corresponds to indistinguishability by formulas with modal depth up to k: For a formula φ in any of the logics considered in this paper, the *modal depth* of φ, denoted with $md(\varphi)$, is the maximal nesting degree of modal operators (i.e., $\Box$ and $\Diamond$) in φ.

▶ **Definition 2.4.** Let $M_1 = (W_1, R_1, \pi_1)$ and $M_2 = (W_2, R_2, \pi_2)$ be Kripke models. We define inductively what it means for states $w_1 \in W_1$ and $w_2 \in W_2$ to be k-bisimilar, for some $k \in \mathbb{N}$, written as $(M_1, w_1) \rightleftharpoons_k (M_2, w_2)$.

- $(M_1, w_1) \rightleftharpoons_0 (M_2, w_2)$ holds if for each propositional variable p, we have that $M_1, w_1 \models p$ if and only if $M_2, w_2 \models p$.
- $(M_1, w_1) \rightleftharpoons_{k+1} (M_2, w_2)$ holds if the following three conditions are satisfied:
  1. $(M_1, w_1) \rightleftharpoons_0 (M_2, w_2)$,
  2. for each successor w_1' of w_1 in M_1, there is a successor w_2' of w_2 in M_2 such that $(M_1, w_1') \rightleftharpoons_k (M_2, w_2')$ (*forward* condition),
  3. for each successor w_2' of w_2 in M_2, there is a successor w_1' of w_1 in M_1 such that $(M_1, w_1') \rightleftharpoons_k (M_2, w_2')$ (*backward* condition).

Full bisimilarity is defined similarly: Pointed models (M_1, w_1) and (M_2, w_2) are *bisimilar* if there is a relation $Z \subseteq W_1 \times W_2$ such that $(w_1, w_2) \in Z$, and for all $(w_1, w_2) \in Z$, we have that w_1 and w_2 satisfy the same propositional variables, and for each successor w_1' of w_1 in M_1, there is a successor w_2' of w_2 in M_2 with $(w_1', w_2') \in Z$ (forward condition), and analogously for each successor w_2' of w_2 in M_2, there is a successor w_1' of w_1 in M_1 with $(w_1', w_2') \in Z$ (back condition). In this case we simply say that M_1, w_1 and M_2, w_2 are *bisimilar*. It is easy to see that bisimilarity implies k-bisimilarity for each k.

▶ **Definition 2.5.** Let $M_1 = (W_1, R_1, \pi_1)$ and $M_2 = (W_2, R_2, \pi_2)$ be Kripke models, and let $w_1 \in W_1$, $w_2 \in W_2$. Then (M_1, w_1) and (M_2, w_2) are k-equivalent for some $k \in \mathbb{N}$, written $(M_1, w_1) \equiv_k (M_2, w_2)$ if for each modal formula φ with $md(\varphi) \leq k$, we have that $M_1, w_1 \models \varphi$ if and only if $M_2, w_2 \models \varphi$.

Again, we simply write $w_1 \equiv_k w_2$ if the models M_1 and M_2 are clear from the context. As mentioned above, k-bisimilarity and k-equivalence coincide. The following result is standard (see, e.g., [2]):

▶ **Proposition 2.6.** *Let $M_1 = (W_1, R_1, \pi_1)$ and $M_2 = (W_2, R_2, \pi_2)$ be Kripke models, and let $w_1 \in W_1$, $w_2 \in W_2$. Then $(M_1, w_1) \rightleftharpoons_k (M_2, w_2)$ if and only if $(M_1, w_1) \equiv_k (M_2, w_2)$.*

For MTL and more generally logics with team semantics, the above notion of bisimulation can be lifted to teams. The following definition is a natural adaptation of k-bisimilarity to the team setting:

▶ **Definition 2.7.** Let $M_1 = (W_1, R_1, \pi_1)$ and $M_2 = (W_2, R_2, \pi_2)$ be Kripke models, let T_1 and T_2 be teams of M_1 and M_2. Then (M_1, T_1) and (M_2, T_2) are k-bisimilar, written as $M_1, T_1 \rightleftharpoons_k M_2, T_2$ if the following holds:

- for each $w_1 \in T_1$, there is some $w_2 \in T_2$ such that $(M_1, w_1) \rightleftharpoons_k (M_2, w_2)$,
- for each $w_2 \in T_2$, there is some $w_1 \in T_1$ such that $(M_1, w_1) \rightleftharpoons_k (M_2, w_2)$.

Full bisimilarity on the team level is defined analogously. In this case we again say that (M_1, T_1) and (M_2, T_2) are bisimilar, and write $M_1, T_1 \rightleftharpoons M_2, T_2$, if there is a relation $Z \subseteq W_1 \times W_2$ satisfying the forward and backward conditions as above, and which additionally satisfies that for each $w_1 \in T_1$, there is some $w_2 \in T_2$ with $(w_1, w_2) \in Z$, and for each $w_2 \in T_2$, there is some $w_1 \in T_1$ with $(w_1, w_2) \in Z$. This notion of team-bisimilarity was first introduced in [11] and [10]. The following result is easily proved by induction on the formula length:

▶ **Proposition 2.8.** *caption*
Let M_1 and M_2 be Kripke models, let T_1 and T_2 be teams of M_1 and of M_2. Then

1. *If $(M_1, T_1) \rightleftharpoons_k (M_2, T_2)$, then for each formula $\varphi \in$ MTL with $md(\varphi) \leq k$, we have that $M_1, T_1 \models \varphi$ if and only if $M_2, T_2 \models \varphi$.*
2. *If $(M_1, T_1) \rightleftharpoons (M_2, T_2)$, then for each formula $\varphi \in$ MTL, we have that $M_1, T_1 \models \varphi$ if and only if $M_2, T_2 \models \varphi$.*

The proof is a straight-forward adaptation of the one in [11].

The expressive power of classical modal logic (i.e., without team semantics) can be characterized by bisimulations. In particular, for every pointed model (M, w), there is a modal formula of modal depth k that exactly characterizes (M, w) up to k-bisimulation.

In the following, we restrict ourselves to a finite set of propositional variables.

3 Main Result: Expressiveness of MTL

In this section, we study the expressive power of MTL. As usual, we measure the expressive power of a logic by the set of properties expressible in it.

▶ **Definition 3.1.** A *team property* is a class of pairs (M, T) where M is a Kripke model and $T \neq \emptyset$ a team of M. For an MTL-formula φ, we say that φ *expresses* the property $\{(M, T) \mid M, T \models \varphi\}$.

Note that most variants of modal dependence logic have the *empty team property*, i.e., for all $\varphi \in$ EMINCL and all Kripke structures M, we have $M, \emptyset \models \varphi$, which obviously does not hold for MTL. However, it immediately follows from the bisimulation invariance of MTL that for every MTL formula φ one of the two possibilities hold:

- For all Kripke structures M, $M, \emptyset \models \varphi$.
- For all Kripke structures M, $M, \emptyset \not\models \varphi$.

For this reason we exclude the empty team in the statement of our results below, but we note that by the remarks above all results cover also the empty team.

▶ **Definition 3.2.** Let P be a team property. Then P is *invariant under k-bisimulation* if for each pair of Kripke models M_1 and M_2 and teams T_1 and T_2 with $(M_1, T_1) \rightleftharpoons_k (M_2, T_2)$ and $(M_1, T_1) \in P$, it follows that $(M_2, T_2) \in P$.

We introduce some (standard) notation. In a model M, the *distance* between two worlds w_1 and w_2 of M is the length of a shortest path from w_1 to w_2 (the distance is infinite if there is no such path). For a world w of a model M and a natural number d, the *d-neighborhood of w in M*, denoted $\mathrm{N}^d_M(w)$, is the set of all worlds w' of M such that the distance from w to w' is at most d. For a team T, with $\mathrm{N}^d_M(T)$ we denote the set $\cup_{w \in T} \mathrm{N}^d_M(w)$. We often identify $\mathrm{N}^d_M(T)$ and the model obtained from M by restriction to the worlds in $\mathrm{N}^d_M(T)$.

▶ **Definition 3.3.** A team property P is *d-local* for some $d \in \mathbb{N}$ if for all models M and teams T, we have

$$(M, T) \in P \text{ if and only if } (\mathrm{N}^d_M(T), T) \in P.$$

We say that P is *local*, if P is d-local for some $d \in \mathbb{N}$.

Since our main result establishes a connection between team properties definable in MTL and team properties definable in first-order logic, we also define what it means for a team property to be expressed by a first-order formula. For a finite set of propositional variables X, we define σ_X as the first-order signature containing a binary relational symbol E (for the edges in our model), a unary relational symbol T (for representing a team), and, for each variable $x \in X$, a unary relational symbol W_x (representing the worlds in which x is true). Kripke models M with teams T (where we only consider variables in X) directly correspond to σ_X structures: A model $M = (W, R, \pi)$ and a team T uniquely determines the σ_X-structure $\mathcal{M}^{\mathrm{FO}}_{M,T}$ with universe W and the obvious interpretations of the symbols in σ_X.

We therefore say that a first-order formula φ over the signature σ_X expresses a team property P, if for all models M with a team T, we have that $(M, T) \in P$ if and only if $\mathcal{M}^{\mathrm{FO}}_{M,T} \models \varphi$. We can now state the main result of this paper:

Table 1 Formulas and sets of formulas used in the proof of Theorem 3.4.

Formula	Intuition	Defined in
$\phi^k_{M,w}$	Characterizes the pointed model (M,w) up to k-bisimilarity	Theorem 3.6
$\Phi^{\rightleftharpoons_k}$	All formulas of the form $\phi^k_{M,w}$ (this is a finite set)	Definition 3.7
$\Phi^{\rightleftharpoons_k}_{M,T}$	Formulas characterizing pointed models (M,w), where $w \in T$, up to k-bisimilarity (this is a finite set)	Definition 3.8
$\varphi^{\rightleftharpoons_k}_{M,T}$	Formula characterizing model M with T up to k-bisimilarity	Definition 3.9

▶ **Theorem 3.4.** *Let P be a team property. Then the following are equivalent:*

(i) *There is an* MTL*-formula which expresses P,*
(ii) *there is a first-order formula which expresses P and P is bisimulation-invariant,*
(iii) *P is invariant under k-bisimulation for some k,*
(iv) *P is bisimulation-invariant and local.*

This result characterizes the expressive power of MTL in several ways. The equivalence of points 1 and 2 is a natural analog to the classic van Benthem theorem which states that standard modal logic directly corresponds to the bisimulation-invariant fragment of first-order logic. It is easy to see that characterizations corresponding to items 3 and 4 also hold in the classical setting. Our result therefore shows that MTL plays the same role for team-based modal logics as ML does for standard modal logic.

The connection between our result and van Benthem's Theorem [23, 24] is also worth discussing. Essentially, van Benthem's Theorem is the same result as ours, where "MTL" is replaced by "ML" and properties of pointed models (i.e., singleton teams) are considered. In ML, classical negation is of course freely available; however the property of a team being a singleton is clearly not invariant under bisimulation—but the property of a team having only one element *up to bisimulation* is. It therefore follows that each property of singleton teams that is invariant under bisimulation and that can be expressed in MTL can already be expressed in ML.

The remainder of Section 3 is devoted to the proof of Theorem 3.4. The proof relies on various formulas that characterize pointed models, teams of pointed models, or team properties up to k-bisimulation, for some $k \in \mathbb{N}$. In Table 1, we summarize the notation used in the following and explain the intuitive meaning of these formulas.

3.1 Expressing Properties in MTL and Hintikka Formulas

We start with a natural characterization of the semantics of splitjunction $\vee$ for ML-formulas.

▶ **Proposition 3.5.** *Let S be a non-empty finite set of* ML*-formulas, let M be a model and T a team. Then $M,T \models \bigvee_{\varphi\in S}\varphi$ if and only if for each world $w \in T$, there is a formula $\varphi \in S$ with $M,\{w\} \models \varphi$.*

The following result is standard:

▶ **Theorem 3.6** ([7, Theorem 32])**.** *For each pointed Kripke model (M,w) and each natural number k, there is a* Hintikka formula *(or characteristic formula) $\phi^k_{M,w} \in$ ML with $md(\phi^k_{M,w}) = k$ such that for each pointed model (M',w'), the following are equivalent:*

1. $M',w' \models \phi^k_{M,w}$,
2. $(M,w) \rightleftharpoons_k (M',w')$.

Clearly, we can choose the Hintikka formulas such that $\phi^k_{M,w}$ is uniquely determined by the bisimilarity type of (M, w). This implies that for k-bisimilar pointed models (M_1, w_1) and (M_2, w_2), the formulas $\phi^k_{M_1,w_1}$ and $\phi^k_{M_2,w_2}$ are identical.

It it clear that Theorem 3.6 does not hold for an infinite set of propositional symbols, since a finite formula can only specify the values of finitely many variables.

We now define the set of all Hintikka formulas that will appear in our later constructions. Informally, $\Phi^{\rightleftharpoons_k}$ is the set of all Hintikka formulas characterizing models up to k-bisimilarity:

▶ **Definition 3.7.** For $k \in \mathbb{N}$, the set $\Phi^{\rightleftharpoons_k}$ is defined as

$$\Phi^{\rightleftharpoons_k} = \left\{\phi^k_{M,w} \mid (M, w) \text{ is a pointed Kripke model}\right\}.$$

An important observation is that $\Phi^{\rightleftharpoons_k}$ is a finite set: This follows since above, we chose the representatives $\phi^k_{M,w}$ to be identical for k-bisimilar pointed models, and since there are only finitely many pointed models up to k-bisimulation. Since $\Phi^{\rightleftharpoons_k}$ is finite, we can in the following freely use disjunctions over arbitrary subsets of $\Phi^{\rightleftharpoons_k}$ and still obtain a finite formula. We will make extensive use of this fact in the remainder of Section 3, often without reference.

Our next definition is used to characterize a team, again up to k-bisimulation. Since teams are sets of worlds, we use sets of formulas to characterize teams in the natural way, by choosing, for each world in the team, one formula that characterizes it.

▶ **Definition 3.8.** For a model M and a team T, let

$$\Phi^{\rightleftharpoons_k}_{M,T} = \{\varphi \in \Phi^{\rightleftharpoons_k} \mid \text{there is some } w \in T \text{ with } M, w \models \varphi\}.$$

Since $\Phi^{\rightleftharpoons_k}_{M,T} \subseteq \Phi^{\rightleftharpoons_k}$, it follows that $\Phi^{\rightleftharpoons_k}_{M,T}$ is finite as well. In fact, it is easy to see that $\left|\Phi^{\rightleftharpoons_k}_{M,T}\right|$ is exactly the number of k-bisimilarity types in T, i.e., the size of a maximal subset of T containing only worlds such that the resulting pointed models are pairwise non-k-bisimilar.

We now combine the formulas from $\Phi^{\rightleftharpoons_k}_{M,T}$ to be able to characterize M and T (up to k-bisimulation) by a single formula:

▶ **Definition 3.9.** For a model M with a team $T \neq \emptyset$, let

$$\varphi^{\rightleftharpoons_k}_{M,T} = \left(\bigwedge_{\varphi \in \Phi^{\rightleftharpoons_k}_{M,T}} \mathsf{E}\varphi\right) \wedge \left(\bigvee_{\varphi \in \Phi^{\rightleftharpoons_k}_{M,T}} \varphi\right).$$

Intuitively, the formula $\varphi^{\rightleftharpoons_k}_{M,T}$ expresses that in a model M' and T' with $M', T' \models \varphi^{\rightleftharpoons_k}_{M,T}$, for each world $w \in T$ there must be some $w' \in T'$ such that $(M, w) \rightleftharpoons_k (M', w')$, and conversely, for each $w' \in T'$, there must be some $w \in T$ with $(M, w) \rightleftharpoons_k (M', w')$, which then implies that (M, T) and (M', T') are indeed k-bisimilar.

From the above, it follows that $\varphi^{\rightleftharpoons_k}_{M,T}$ is a finite MTL-formula. Therefore, with the above intuition, it follows that $\varphi^{\rightleftharpoons_k}_{M,T}$ expresses k-bisimilarity with (M, T).

▶ **Proposition 3.10.** *Let M_1, M_2 be Kripke models with teams nonempty T_1, T_2. Then the following are equivalent:*

- $(M_1, T_1) \rightleftharpoons_k (M_2, T_2)$
- $M_1, T_1 \models \varphi^{\rightleftharpoons_k}_{M_2,T_2}$.

3.2 Proof of Theorem 3.4

In this section, we prove our main result, Theorem 3.4.

3.2.1 Proof of equivalence 3.4.(1) ↔ 3.4.(3)

Proof. The direction $1 \to 3$ follows immediately from Proposition 2.8. For the converse, assume that P is invariant under k-bisimulation. Without loss of generality assume $P \neq \emptyset$. We claim that the formula

$$\varphi_P := \mathbin{⫿}_{(M,T)\in P} \varphi^{\rightleftharpoons^k}_{M,T}$$

expresses P.

First note that φ_P can be written as the disjunction of only finitely many formulas: Each $\varphi^{\rightleftharpoons^k}_{M,T}$ is uniquely defined by a subset of the finite set $\Phi^{\rightleftharpoons^k}$, therefore there are only finitely many formulas of the form $\varphi^{\rightleftharpoons^k}_{M,T}$.

We now show that for each model M and team T, we have that $(M,T) \in P$ if and only if $M,T \models \varphi_P$. First assume that $(M,T) \in P$. Then the fact that $(M,w) \rightleftharpoons^k (M,w)$ for each model M, each world w and each number k and Proposition 3.10 imply that $M,T \models \varphi^{\rightleftharpoons^k}_{M,T}$. Therefore, $M,T \models \varphi_P$. For the converse, assume that $M,T \models \varphi_P$. Then there is some $(M',T') \in P$ with $M,T \models \varphi^{\rightleftharpoons^k}_{M',T'}$. Due to Proposition 3.10, it follows that $(M,T) \rightleftharpoons^k (M',T')$. Since P is invariant under k-bisimulation, it follows that $(M,T) \in P$ as required. ◀

3.2.2 Proof of implication 3.4.(3) → 3.4.(2)

Proof. It suffices to show that P can be expressed in first-order logic. This follows using essentially the standard translation from modal into first-order logic. Since classical disjunction is of course available in first-order logic, the proof of the implication $3 \to 1$ shows that it suffices to express each $\varphi^{\rightleftharpoons^k}_{M,T}$ (expressing team-bisimilarity to M,T) in first-order logic.

Each of the Hintikka formulas $\phi^k_{M,w}$ (expressing bisimilarity to the pointed model M,w) is a standard modal formula, therefore an application of the standard translation gives a first-order formula $\phi^{k,\mathrm{FO}}_{M,w}$ with a free variable x such that for all models M' and worlds w', we have that $M',w' \models \phi^k_{M,w}$ if and only if $\mathcal{M}^{\mathrm{FO}}_{M',\emptyset} \models \phi^{k,\mathrm{FO}}_{M,w}(w)$. We now show how to express $\varphi^{\rightleftharpoons^k}_{M,T}$ (expressing team-bisimilarity to M,T) in first-order logic.

Recall that $\varphi^{\rightleftharpoons^k}_{M,T}$ is defined as $\left(\bigwedge_{\varphi\in\Phi^{\rightleftharpoons^k}_{M,T}} \mathsf{E}\varphi\right) \wedge \left(\bigvee_{\varphi\in\Phi^{\rightleftharpoons^k}_{M,T}} \varphi\right)$. Therefore, a first-order representation of $\varphi^{\rightleftharpoons^k}_{M,T}$ is given as

$$\left(\bigwedge_{\varphi\in\Phi^{\rightleftharpoons^k}_{M,T}} \exists w(T(w) \wedge \varphi^{\mathrm{FO}}(w))\right) \wedge \left(\forall w(T(w) \implies \bigvee_{\varphi\in\Phi^{\rightleftharpoons^k}_{M,T}} \varphi^{\mathrm{FO}}(w)\right),$$

where φ^{FO} is the standard translation of φ into first-order logic as mentioned above. This concludes the proof. ◀

3.2.3 Proof of implication 3.4.(2) → 3.4.(4)

Proof. Let φ be the first-order formula expressing P. Since φ is first-order, we know that φ is Hanf-local. Let d be the Hanf-locality rank of φ. We show that φ is $2d$-local. Therefore, let M be a model with team T. We show that $\mathcal{M}^{\mathrm{FO}}_{M,T} \models \varphi$ if and only if $\mathcal{M}^{\mathrm{FO}}_{\mathrm{N}^{2d}_M(T),T} \models \varphi$. Since φ is bisimulation-invariant, it suffices to construct models M_1 and M_2 containing T such that

- (M_1,T) and (M,T) are team-bisimilar,
- (M_2,T) and $(\mathrm{N}^{2d}_M(T),T)$ are team-bisimilar,
- $\mathcal{M}^{\mathrm{FO}}_{M_1,T} \models \varphi$ if and only if $\mathcal{M}^{\mathrm{FO}}_{M_2,T} \models \varphi$.

We first define M^{DISS} as the model obtained from M by disconnecting $\mathrm{N}^{2d}_M(T)$ from the remainder of the model, i.e., by removing all edges between $\mathrm{N}^{2d}_M(T)$ and $M \setminus \mathrm{N}^{2d}_M(T)$. Since M^{DISS} is also obtained from $\mathrm{N}^{2d}_M(T)$ by adding the remainder of the model M without connecting the added worlds to $\mathrm{N}^{2d}_M(T)$, it is obvious that $(M^{\mathrm{DISS}}, T) \rightleftharpoons (\mathrm{N}^{2d}_M(T), T)$. We now define the models M_1 and M_2 such that $(M_1, T) \rightleftharpoons (M, T)$ and $(M_2, T) \rightleftharpoons (M^{\mathrm{DISS}}, T)$ (and hence $(M_2, T) \rightleftharpoons (\mathrm{N}^{2d}_M(T), T)$) as follows:

- M_1 and M_2 are obtained from M and M^{DISS} by adding the exact same components: For each $w \in M$ (note that M and M^{DISS} have the exact same set of worlds), countably infinitely many copies of $\mathrm{N}^{2d}_M(w)$ and of $\mathrm{N}^{2d}_{M^{\mathrm{DISS}}}(w)$ are added to both M_1 and M_2.
- for $n \in \mathbb{N}$, and $i \in \{1, 2\}$, with $C^{\mathrm{DISS}}_{i,n}(w)$, we denote the n-th copy of $\mathrm{N}^{2d}_{M^{\mathrm{DISS}}}(w)$ in M_i, the *center* of $C^{\mathrm{DISS}}_{i,n}(w)$ is the copy of w in $C^{\mathrm{DISS}}_{i,n}(w)$.
- for $n \in \mathbb{N}$, and $i \in \{1, 2\}$, with $C^{\mathrm{CONN}}_{i,n}(w)$, we denote the n-th copy of $\mathrm{N}^{2d}_M(w)$ in M_i, the *center* of $C^{\mathrm{CONN}}_{i,n}(w)$ is the copy of w in $C^{\mathrm{CONN}}_{i,n}(w)$.

In the above, when we "copy" a part of a (Kripke) model, this includes copying the values of the involved propositional variables in these worlds (this is reflected in the resulting first-order models in the obvious way). However, we stress that the team T is treated differently: The set T is not enlarged with the copy operation, i.e., a copy of a world in T is itself not an element of T.

Since M_1 and M_2 are obtained from M and M^{DISS} by adding new components that are not connected to the original models, it clearly follows that (M, T) and (M_1, T) are team-bisimilar, and (M^{DISS}, T) and (M_2, T) are team-bisimilar. Note that each w in the M-part of M_1 is the center of a $2d$-environment isomorphic to $C^{\mathrm{CONN}}_{2,n}(w)$, and each w in the M^{DISS}-part of M_2 is the center of a $2d$-environment isomorphis to $C^{\mathrm{DISS}}_{1,n}(w)$.

Since the models M (M^{DISS}) contain one copy of each $\mathrm{N}^{2d}_M(w)$ ($\mathrm{N}^{2d}_{M^{\mathrm{DISS}}}(w)$), both M_1 and M_2 contain countably infinitely many copies of each $\mathrm{N}^{2d}_M(w)$ and each $\mathrm{N}^{2d}_{M^{\mathrm{DISS}}}(w)$. Let S_1 be the subset of M_1 containing only the points from the M-part of M_1, plus the center of each $C^{\mathrm{CONN}}_{1,n}(w)$, and the center of each $C^{\mathrm{DISS}}_{1,n}(w)$. Similarly, let S_2 be the subset of M_2 containing only the points from the M^{DISS}-part of M_2 plus the centers of the added components.

Since M_1 and M_2 contain the same number of copies of each relevant neighborhood, there is a bijection $f\colon S_1 \to S_2$ such that for each $w \in S_1$, the $2d$-neighborhoods of w and $f(w)$ are isomorphic. Now f can be modified such that for each $w \in M$ which has distance at most d to a world in T, the value $f(w)$ is the corresponding world in the M^{DISS}-part of M_2. The thus-modified f now satisfies that for each $w \in S_1$, the d-neighborhoods of w and $f(w)$ are isomorphic. We can easily extend f to worlds in $C^{\mathrm{DISS}}_{1,n}$ and $C^{\mathrm{CONN}}_{1,n}$ that are not the center of their respective components by mapping such a world w in $C^{\mathrm{DISS}}_{1,n}$ to the copy of w in $C^{\mathrm{DISS}}_{2,n}$, and analogously for $C^{\mathrm{CONN}}_{1,n}$.

Therefore, we have constructed a bijection $f\colon M_1 \to M_2$ such that for each $w \in M_1$, the d-neighborhood of w in M_1 is isomorphic to the d-neighborhood of w in M_2. Since φ is Hanf-local with rank d, this implies that $\mathcal{M}^{\mathrm{FO}}_{M_1,T} \models \varphi$ if and only if $\mathcal{M}^{\mathrm{FO}}_{M_2,T} \models \varphi$, as required. ◀

The proof of this implication uses ideas from Otto's proof of van Benthem's classical theorem presented in [17]. However our proof is based on the Hanf-locality of first-order expressible properties, whereas Otto's proof uses Ehrenfeucht-Fraïssé games, as a consequence, our construction requires an infinite number of copies of each model due to cardinality reasons.

3.2.4 Proof of implication 3.4.(4) → 3.4.(3)

Proof. Assume that P is invariant under bisimulation, and P is k-local for some $k \in \mathbb{N}$. We show that P is invariant under k-bisimulation. Hence let $M_1, T_1 \rightleftharpoons_k M_2, T_2$. Since P is invariant under bisimulation, we can without loss of generality assume that M_1 and M_2 are directed forests, that M_1 contains only worlds connected to worlds in T_1, and analogously for M_2 and T_2. Since P is also k-local, we can also assume that M_1 contains no world with a distance of more than k to T_1, and analogously for M_2 and T_2. From these assumptions, it immediately follows that $M_1, T_1 \rightleftharpoons M_2, T_2$, and, since P is invariant under bisimulation, this implies that $(M_1, T_1) \in P$ if and only if (M_2, T_2) in P, as required. ◀

4 Alternative logical characterisations for the bisimulation invariant properties

Research on variants of (modal) dependence logic has concentrated on logics defined in terms of independence and inclusion atoms. Analogously to MDL, these logics are invariant under bisimulation but are strictly less expressive than MTL [11]. On the other hand, extended modal dependence logic, EMDL, uses dependence atoms but allows them to be applied to ML-formulas instead of just proposition symbols [4]. This variant is also known to be a proper sub-logic of MTL being able to express all downwards-closed properties that are invariant under k-bisimulation for some $k \in \mathbb{N}$, and equivalent to ML($\mathbin{⫽}$) [10].

In this section we systematically study the expressive power of variants of EMDL replacing dependence atoms by independence and inclusion atoms. Depending on whether we also allow classical disjunction or not, this gives four logics, namely EMIL (Extended Modal Independence Logic), EMIL($\mathbin{⫽}$) (EMIL extended with classical disjunction), EMINCL (Extended Modal Inclusion Logic) and EMINCL($\mathbin{⫽}$) (EMINCL extended with classical disjunction). We study the expressiveness of these logics, and show that while EMINCL($\mathbin{⫽}$) is as expressive as MTL, for each of the other three logics there is an MTL-expressible property that cannot be expressed in the logic. In the last section, we also study the extension of ML by first-order definable generalised dependence atoms, and show that the resulting logic—even without the addition of classical disjunction—is equivalent to MTL.

4.1 Extended Modal Independence Logic (EMIL)

We first consider *Extended Modal Independence Logic* (EMIL). Syntactically, EMIL extends ML by the following: If $\overline{P}$, $\overline{Q}$, and $\overline{R}$ are finite sets of ML-formulas, then $\overline{P} \perp_{\overline{R}} \overline{Q}$ is an EMIL-formula. The semantics of this *extended independence atom* are defined by lifting the definition for propositional variables given in [11] to ML-formulas as follows.

For a formula φ and a world w, we write $\varphi(w)$ for the function defined as $\varphi(w) = 1$ if $M, \{w\} \models \varphi$, and $\varphi(w) = 0$ otherwise (the model M will always be clear from the context). For a set of formulas $\overline{F}$ and worlds w_1, w_2, we write $w_1 \equiv_{\overline{F}} w_2$ if $\varphi(w_1) = \varphi(w_2)$ for each $\varphi \in \overline{F}$.

$$M, T \models \overline{P} \perp_{\overline{R}} \overline{Q} \quad \Leftrightarrow \quad \begin{array}{l} \forall w, w' \in T\colon w \equiv_{\overline{R}} w' \text{ implies } \exists w'' \in T\colon \\ w'' \equiv_{\overline{P}} w \text{ and } w'' \equiv_{\overline{Q}} w' \text{ and } w'' \equiv_{\overline{R}} w. \end{array}$$

The extension of EMIL by classical disjunction $\mathbin{⫽}$ is denoted by EMIL($\mathbin{⫽}$).

We will next show that EMIL($\mathbin{⫽}$) is a proper sub-logic of MTL. The following lemma will be used in the proof.

▶ **Lemma 4.1.** *Let $M = (W, R, \pi)$ be a Kripke model such that $R = \emptyset$ and $T \subseteq W$ a team. Then for all $\varphi \in \mathsf{EMIL}(\mathbin{⫽})$ it holds that if $M, T \models \varphi$, then $M, \{w\} \models \varphi$ for all $w \in T$.*

Proof. A straight-forward induction on the construction of φ using the facts that a singleton team trivially satisfies all independence atoms, and the empty team satisfies all formulas of $\mathsf{EMIL}(\mathbin{⫽})$. ◀

▶ **Theorem 4.2.** $\mathsf{EMDL} \subsetneq \mathsf{EMIL} \subseteq \mathsf{EMIL}(\mathbin{⫽}) \subsetneq \mathsf{MTL}$.

Proof sketch. The first inclusion follows from the fact that dependence atoms can be expressed by independence atoms. The inclusion is strict since EMDL is downwards-closed and EMIL is not. For the last inclusion, note that every property expressible in $\mathsf{EMIL}(\mathbin{⫽})$ is invariant under bisimulation, hence it follows that MTL can express every $\mathsf{EMIL}(\mathbin{⫽})$-expressible property due to Theorem 3.4. For the strictness, note that Lemma 4.1 can be used to show that the property expressed by the MTL formula $\mathsf{E}p$ cannot be expressed in $\mathsf{EMIL}(\mathbin{⫽})$. ◀

4.2 Extended Modal Inclusion Logic

Analogously to EMIL, we now define *Extended Modal Inclusion Logic*, EMINCL. EMINCL extends the syntax of ML with the following rule: If $\varphi_1, \ldots, \varphi_n$ and $\psi_1, \ldots, \psi_n$ are ML-formulas, then $(\varphi_1, \ldots, \varphi_n) \subseteq (\psi_1, \ldots, \psi_n)$ is an EMINCL-formula. The semantics of this *inclusion atom* are lifted from the first-order setting [5] to the extended modal case:

> $M, T \models (\varphi_1, \ldots, \varphi_n) \subseteq (\psi_1, \ldots, \psi_n)$ if for every world $w \in T$ there is a world $w' \in T$ such that $\varphi_i(w) = \psi_i(w')$ for each $i \in \{1, \ldots, n\}$.

The extension of EMINCL by classical disjunction $\mathbin{⫽}$ is denoted by $\mathsf{EMINCL}(\mathbin{⫽})$.

Analogously to first-order inclusion logic [6], the truth of EMINCL-formulas is preserved under unions of teams. Hence we get the following result.

▶ **Theorem 4.3.** EMINCL *is strictly less expressive than* MTL.

Next we want to show that $\mathsf{EMINCL}(\mathbin{⫽})$ is as powerful as MTL.

▶ **Theorem 4.4.** *Let P be a team property. Then the following are equivalent:*
1. *P is invariant under k-bisimulation.*
2. *There is an $\mathsf{EMINCL}(\mathbin{⫽})$-formula φ with $md(\varphi) = k$ that characterizes P.*

Proof. The direction from 2 to 1 follows by a straight-forward extension of the proof of Proposition 2.8. For the converse, assume that P is invariant under k-bisimulation. From the proof of Theorem 3.4, we know that it suffices to construct an $\mathsf{EMINCL}(\mathbin{⫽})$-formula φ that is equivalent to the MTL-formula $\mathbin{⫽}_{(M,T)\in P} \varphi^{\Rrightarrow k}_{M,T}$. Since the $\mathbin{⫽}$-operator is available in $\mathsf{EMINCL}(\mathbin{⫽})$, it suffices to show how to express the formula $\varphi^{\Rrightarrow k}_{M,T}$ for each model M and team T as an $\mathsf{EMINCL}(\mathbin{⫽})$-formula. Recall that

$$\varphi^{\Rrightarrow k}_{M,T} = \left(\bigwedge_{\varphi \in \Phi^{\Rrightarrow k}_{M,T}} \mathsf{E}\varphi \right) \wedge \left(\bigvee_{\varphi \in \Phi^{\Rrightarrow k}_{M,T}} \varphi \right).$$

The second conjunct already is an $\mathsf{EMINCL}(\mathbin{⫽})$-formula, hence it suffices to show how $\mathsf{E}\varphi$ can be expressed for an ML-formula φ. As discussed earlier, $M, T \models \mathsf{E}\varphi$ for an ML-formula φ if and only if there is a world $w \in T$ with $M, \{w\} \models \varphi$. Hence from the semantics of the inclusion atom, it is clear that $\mathsf{E}\varphi$ is equivalent to $(x \vee \neg x) \subseteq (\varphi)$. This concludes the proof. ◀

4.3 ML with FO-definable generalized dependence atoms

In this section we show that MTL, and the bisimulation invariant properties, can be captured as the extension of ML by all generalized dependence atoms definable in first-order logic without identity. The notion of a generalized dependence atom in the modal context was introduced in [11]. A closely related notion was introduced and studied in the first-order context in [13]. The semantics of a generalized dependence atom D is determined essentially by a property of teams.

In the following we are interested in generalized dependence atoms definable by first-order formulae, defined as follows: Suppose that D is an atom of width n, that is, an atom that applies to n propositional variables (for example the atom in (2)). We say that D is FO-definable if there exists a FO-sentence ϕ over signature $\langle A_1, \ldots, A_n \rangle$ such that for all Kripke models $M = (W, R, \pi)$ and teams T,

$$M, T \models D(p_1, \ldots, p_n) \iff \mathcal{A} \models \phi,$$

where $\mathcal{A}$ is the first-order structure with universe T and relations $A_i^{\mathcal{A}}$ for $1 \leq i \leq n$, where for all $w \in T$, $w \in A_i^{\mathcal{A}} \Leftrightarrow p_i \in \pi(w)$.

In our "extended" setting the arguments to a generalized dependence atom $D(\varphi_1, \ldots, \varphi_n)$ can be arbitrary ML-formulas instead of propositional variables. Hence the relation A_i is now interpreted by the worlds of T in which φ_i is satisfied. We denote by $\mathsf{ML}^{\mathsf{FO}}$ the extension of ML by all generalized dependence atoms D that are FO-definable without identity.

▶ **Theorem 4.5.** $\mathsf{ML}^{\mathsf{FO}}$ *is equally expressive as* MTL.

Proof. In the proof of Theorem 6.8 in [11] it is showed that $\mathsf{ML}^{\mathsf{FO}}$ is invariant under bisimulation in the case where generalised atoms may be applied only to propositional variables. The proof easily extends to the setting where arbitrary ML-formulas may appear as arguments to a generalised dependence atom. Therefore, $\mathsf{ML}^{\mathsf{FO}}$ is not more expressive than MTL. For the converse, let P be a property that can be expressed in MTL. From Theorem 3.4, it follows that P is invariant under k-bisimulation, and from the proof of Theorem 3.4, we know that it suffices to express the formula $\mathbb{V}_{(M,T)\in P} \varphi^{\rightrightarrows k}_{M,T}$ in $\mathsf{ML}^{\mathsf{FO}}$. We can do this with the following first-order definable atom (by suitably choosing the parameters $n, m \in \mathbb{N}$):

> $M, T \models D(\varphi_1^1, \ldots, \varphi_n^1, \varphi_1^2, \ldots, \varphi_n^2, \ldots, \varphi_1^m, \ldots, \varphi_n^m)$ if and only if there is some $k \in \{1, \ldots, m\}$ such that each $w \in T$ satisfies some φ_i^k, and for each $j \in \{1, \ldots, n\}$, there is some $w \in T$ that satisfies φ_j^k.

The atom D can now be FO-defined by replacing the exists/for all quantifiers on the indices with disjunctions/conjunctions:

$$\bigvee_{k\in\{1,\ldots,m\}} \left(\forall x \left(A_1^k(x) \vee \cdots \vee A_n^k(x) \right) \wedge \bigwedge_{j\in\{1,\ldots,n\}} (\exists x\ A_j^k(x)) \right)$$

Then, the atom D applied to the formulas in $\varphi^{\rightrightarrows k}_{M,T}$ for all $(M, T) \in P$ gives a formula expressing P. ◀

5 Conclusion

Our results show that, with respect to expressive power, modal team logic is a natural upper bound for all the logics studied so far in the area of modal team semantics. Overall, an

interesting picture of the characterization of the expressiveness of modal logics in terms of bisimulation emerges: Let us say that "invariant under bounded bisimulation" means invariant under k-bisimulation for some finite k. Then we have the following hierarchy of logics:

- Due to van Benthem's theorem [24], ML can exactly express all properties of pointed models that are FO-definable and invariant under bisimulation.
- Due to [10], ML with team semantics and extended with classical disjunction $\veebar$ can exactly express all properties of teams that are invariant under bounded bisimulation and additionally downwards-closed.
- Our result shows that ML with team semantics and extended with classical negation $\sim$ can exactly express all properties of teams that are invariant under bounded bisimulation.

A number of open questions in the realm of modal logics with team semantics remain:

1. In the proof of Theorem 4.5, for each k, there is only a finite width of the D-operator above needed to express all properties that are invariant under k-bisimulation. However, the theorem leaves open the question whether there is a "natural" atom D or an atom with "restricted width" that gives the entire power of MTL.
2. Can we axiomatize MTL? Axiomatizability of sublogics of MTL has been studied, e.g., in [25] and [19].
3. While we mentioned a number of complexity results on modal dependence logic and some of its extensions, this issue remains unsettled for full MTL. In particular, what is the complexity of satisfiability and validity of MTL?

References

1 Samson Abramsky and Jouko A. Väänänen. From IF to BI. *Synthese*, 167(2):207–230, 2009.

2 Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic*, volume 53 of *Cambridge Tracts in Theoretical Computer Scie*. Cambridge University Press, Cambridge, 2001.

3 J. Ebbing and P. Lohmann. Complexity of model checking for modal dependence logic. In *Theory and Practice of Computer Science (SOFSEM)*, number 7147 in Lecture Notes in Computer Science, pages 226–237, Berlin Heidelberg, 2012. Springer Verlag.

4 Johannes Ebbing, Lauri Hella, Arne Meier, Julian-Steffen Müller, Jonni Virtema, and Heribert Vollmer. Extended modal dependence logic. In Leonid Libkin, Ulrich Kohlenbach, and Ruy J. G. B. de Queiroz, editors, *WoLLIC*, volume 8071 of *Lecture Notes in Computer Science*, pages 126–137. Springer, 2013.

5 Pietro Galliani. Inclusion and exclusion dependencies in team semantics – on some logics of imperfect information. *Ann. Pure Appl. Logic*, 163(1):68–84, 2012.

6 Pietro Galliani and Lauri Hella. Inclusion logic and fixed point logic. In Simona Ronchi Della Rocca, editor, *CSL*, volume 23 of *LIPIcs*, pages 281–295. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2013.

7 Valentin. Goranko and Martin Otto. *Handbook of Modal Logic*, chapter Model Theory of Modal Logic, pages 255–325. Elsevier, 2006.

8 Rajeev Goré, Barteld P. Kooi, and Agi Kurucz, editors. *Advances in Modal Logic 10, invited and contributed papers from the tenth conference on "Advances in Modal Logic," held in Groningen, The Netherlands, August 5-8, 2014*. College Publications, 2014.

9 Erich Grädel and Jouko Väänänen. Dependence and independence. *Studia Logica*, 101(2):399–410, 2013.

10 Lauri Hella, Kerkko Luosto, Katsuhiko Sano, and Jonni Virtema. The expressive power of modal dependence logic. In Goré et al. [8], pages 294–312.

11 Juha Kontinen, Julian-Steffen Müller, Henning Schnoor, and Heribert Vollmer. Modal independence logic. In Goré et al. [8], pages 353–372.

12 Juha Kontinen, Julian-Steffen Müller, Henning Schnoor, and Heribert Vollmer. A van benthem theorem for modal team semantics. *CoRR*, abs/1410.6648, 2014.

13 Antti Kuusisto. A double team semantics for generalized quantifiers. *Journal of Logic, Language and Information*, 24(2):149–191, 2015.

14 Peter Lohmann and Heribert Vollmer. Complexity results for modal dependence logic. *Studia Logica*, 101(2):343–366, 2013.

15 Julian-Steffen Müller. Complexity of model checking in modal team logic. *Computation in Europe*, 2012.

16 Julian-Steffen Müller. *Satisfiability and Model Checking in Team Based Logics*. PhD thesis, Leibniz Universität Hannover, 2014.

17 M. Otto. Elementary proof of the van Benthem-Rosen characterisation theorem. Technical Report 2342, Darmstadt University of Technology, 2004.

18 David Michael Ritchie Park. Concurrency and automata on infinite sequences. In Peter Deussen, editor, *Theoretical Computer Science, 5th GI-Conference, Karlsruhe, Germany, March 23-25, 1981, Proceedings*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer, 1981.

19 Katsuhiko Sano and Jonni Virtema. Axiomatizing propositional dependence logics. *CoRR*, abs/1410.5038, 2014.

20 Merlijn Sevenster. Model-theoretic and computational properties of modal dependence logic. *J. Log. Comput.*, 19(6):1157–1173, 2009.

21 Jouko Väänänen. *Dependence Logic*. Cambridge University Press, 2007.

22 Jouko Väänänen. Modal dependence logic. *New Perspectives on Games and Interaction*, 5:237–254, 2009.

23 Johan van Benthem. *Modal Correspondence Theory*. PhD thesis, Universiteit van Amsterdam, Instituut voor Logica en Grondslagenonderzoek van de Exacte Wetenschappen, 1977.

24 Johan van Benthem. *Modal Logic and Classical Logic*. Bibliopolis, 1985.

25 Fan Yang. *On Extensions and Variants of Dependence Logic*. PhD thesis, University of Helsinki, 2014.

Axiomatizing Propositional Dependence Logics*

Katsuhiko Sano[1] and Jonni Virtema[1,2,3]

1 **Japan Advanced Institute of Science and Technology, Japan**
2 **Leibniz Universität Hannover, Germany**
3 **University of Tampere, Finland**
{katsuhiko.sano,jonni.virtema}@gmail.com

Abstract

We give sound and complete Hilbert-style axiomatizations for propositional dependence logic ($\mathcal{PD}$), modal dependence logic ($\mathcal{MDL}$), and extended modal dependence logic ($\mathcal{EMDL}$) by extending existing axiomatizations for propositional logic and modal logic. In addition, we give novel labeled tableau calculi for $\mathcal{PD}$, $\mathcal{MDL}$, and $\mathcal{EMDL}$. We prove soundness, completeness and termination for each of the labeled calculi.

1998 ACM Subject Classification F.4.1 Mathematical logic

Keywords and phrases propositional dependence logic, modal dependence logic, axiomatization, tableau calculus

Digital Object Identifier 10.4230/LIPIcs.CSL.2015.292

1 Introduction

Functional dependences occur everywhere in science, e.g., in descriptions of discrete systems, in database theory, social choice theory, mathematics, and physics. Modal logic is an important formalism utilized in the research of numerous disciplines including many of the fields mentioned above. With the aim to express functional dependences in the framework of logic Väänänen [9] introduced *dependence logic*. Dependence logic extends first-order logic with novel atomic formulae called *dependence atoms*. The intuitive meaning of the first-order dependence atom $=(t_1, \ldots, t_n)$ is that the value of the term t_n is functionally determined by the values of the terms $t_1, \ldots, t_{n-1}$. With the aim to express functional dependences in the framework of modal logic, Väänänen [10] introduced *modal dependence logic* ($\mathcal{MDL}$). Modal dependence logic extends modal logic with *propositional dependence atoms*. A propositional dependence atom $\mathrm{dep}(p_1, \ldots, p_n, q)$ intuitively states that the truth value of the proposition q is functionally determined by the truth values of the propositions $p_1, \ldots, p_n$. It was soon realized that $\mathcal{MDL}$ lacks the ability to express temporal dependencies; there is no mechanism in $\mathcal{MDL}$ to express dependencies that occur between different points of the model. This is due to the restriction that only proposition symbols are allowed in the dependence atoms of modal dependence logic. To overcome this defect Ebbing et al. [1] introduced *extended modal dependence logic* ($\mathcal{EMDL}$) by extending the scope of dependence atoms to arbitrary modal formulae. Dependence atoms in extended modal dependence logic are of the form $\mathrm{dep}(\varphi_1, \ldots \varphi_n, \psi)$, where $\varphi_1, \ldots, \varphi_n, \psi$ are formulae of modal logic.

* The work of the first author was partially supported by JSPS KAKENHI Grant-in-Aid for Young Scientists (B) Grant Number 15K21025 and by JSPS Core-to-Core Program (A. Advanced Research Networks). The work of the second author was supported by grant 266260 of the Academy of Finland, and by Jenny and Antti Wihuri Foundation.

In recent years the research around modal dependence logic has been active. The focus has been in the computational complexity and in the expressive powers of related formalisms. Sevenster [8] proved that the satisfiability problem for modal dependence logic is NEXPTIME-complete, whereas Ebbing and Lohmann [2] showed that the related model checking problem is NP-complete. Ebbing et al. [1] extended these results to handle also $\mathcal{EMDL}$. Subsequently Virtema [11] showed that the validity problems for $\mathcal{MDL}$ and $\mathcal{EMDL}$ are NEXPTIME-hard and contained in $\mathsf{NEXPTIME}^{\mathsf{NP}}$. Moreover he showed that the corresponding problem for the propositional fragment $\mathcal{PD}$ (see Section 2.1 for a definition) of $\mathcal{MDL}$ is NEXPTIME-complete.

Hella et al. [4] gave *a van Benthem–style characterization* of the expressive power of $\mathcal{EMDL}$ via the so-called *team k-bisimulation*. In the article it was also shown that the expressive powers of $\mathcal{EMDL}$ and $\mathcal{ML}(\mathbin{\mathbb{V}})$ (modal logic extended with intuitionistic disjunction) coincide. More recently Kontinen et al. (in the manuscript [6]) gave another van Benthem–style characterization for the expressive power of the so-called *modal team logic*. Moreover, in the manuscript [7], Sano and Virtema gave a Goldblatt–Thomason theorem for $\mathcal{MDL}$ and $\mathcal{EMDL}$. They also showed that with respect to frame definability $\mathcal{MDL}$ and $\mathcal{EMDL}$ coincide with a fragment of modal logic extended with the universal modality in which the universal modality occurs only positively. These characterization truly demonstrate the naturality of the related languages.

In this paper we give sound and complete axiomatizations for variants of propositional and modal dependence logics ($\mathcal{PD}$, $\mathcal{PL}(\mathbin{\mathbb{V}})$, $\mathcal{MDL}$, $\mathcal{EMDL}$, and $\mathcal{ML}(\mathbin{\mathbb{V}})$). We give Hilbert-style axiomatizations for these logics by extending existing axiomatizations for propositional logic and modal logic. In addition, we give novel labeled tableau calculi for these logics. This paper is one of the first articles on proof theory of propositional and modal dependence logics. The only other work known by the authors of this article is the PhD thesis of Fan Yang [12] and the subsequent manuscript [13]. Among other things, in her thesis, Yang presents axiomatizations of variants of propositional dependence logic based on natural deduction. Our Hilbert style axiomatization of $\mathcal{PD}$ coincides in essence with the natural deduction system given by Yang. However our axiomatization avoids the complexity of the system of Yang by concentrating on the proof-theoretic essence of the axiomatization. Provided that a Hilbert-style axiomatization for the negation normal form fragment of propositional logic is given, we specify *one* inference rule which gives us an axiomatization of $\mathcal{PD}$.

The article is structured as follows. In Section 2 we introduce the required notions and definitions. In Section 3 we give Hilbert-style axiomatizations for propositional and modal dependence logics. In Section 4 we present labeled tableau calculi for these logics.

2 Preliminaries

The syntax of propositional logic ($\mathcal{PL}$) and modal logic ($\mathcal{ML}$) could be defined in any standard way. However, when we consider extensions of $\mathcal{PL}$ and $\mathcal{ML}$ by dependence atoms, it is useful to assume that all formulae are in *negation normal form*, i.e., negations occur only in front of atomic propositions. Thus we will define the syntax of $\mathcal{PL}$ and $\mathcal{ML}$ in negation normal form. When φ is a formula of $\mathcal{PL}$ or $\mathcal{ML}$, we denote by $\varphi^{\perp}$ the equivalent formula that is obtained from $\neg\varphi$ by pushing all negations to the atomic level. Furthermore, we define $\varphi^{\top} := \varphi$. When $\vec{a}$ is a tuple of symbols of length k, we denote by a_j the jth element of $\vec{a}$, $j \leq k$. When φ is a formula, $|\varphi|$ denotes the number of symbols in φ excluding negations and brackets. When A is a set $|A|$ denotes the number of elements in A. When $f : A \to B$ is a function and $C \subseteq A$, we define $f[C] := \{f(a) \mid a \in C\}$.

2.1 Propositional logic with team semantics

Let $\mathsf{PROP} = \{z_i \mid i \in \mathbb{N}\}$ denote the set of exactly all *propositional variables*, i.e., *proposition symbols*. We mainly use metavariables p, q, p_1, p_2, q_1, q_2, etc., in order to refer to the variable symbols in PROP. Let D be a finite, possibly empty, subset of PROP. A function $s : D \to \{0,1\}$ is called an *assignment*. A set X of assignments $s : D \to \{0,1\}$ is called a *propositional team*. The set D is the *domain* of X. Note that the empty team $\emptyset$ does not have a unique domain; any subset of PROP is a domain of the empty team. By $\{0,1\}^D$, we denote the set of all assignments $s : D \to \{0,1\}$.

Let Φ be a set of proposition symbols. The set of formulae for propostional logic $\mathcal{PL}(\Phi)$ is generated by the grammar: $\varphi ::= p \mid \neg p \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi)$, where $p \in \Phi$.

By $\models_{\mathcal{PL}}$, we denote the ordinary satisfaction relation of propositional logic defined via assignments. Next we define the team semantics of propositional logic.

▶ **Definition 1.** Let Φ be a set of atomic propositions and let X be a propositional team. The satisfaction relation $X \models \varphi$ for $\mathcal{PL}(\Phi)$ is defined as follows. Note that, we always assume that the proposition symbols that occur in φ are also in the domain of X.

$$\begin{aligned}
X \models p &\quad\Leftrightarrow\quad \forall s \in X : s(p) = 1.\\
X \models \neg p &\quad\Leftrightarrow\quad \forall s \in X : s(p) = 0.\\
X \models (\varphi \wedge \psi) &\quad\Leftrightarrow\quad X \models \varphi \text{ and } X \models \psi.\\
X \models (\varphi \vee \psi) &\quad\Leftrightarrow\quad Y \models \varphi \text{ and } Z \models \psi, \text{ for some } Y, Z \text{ such that } Y \cup Z = X.
\end{aligned}$$

▶ **Proposition 2** ([8]). *Let φ be a formula of propositional logic and X a propositional team. Then $X \models \varphi \Leftrightarrow \forall s \in X : s \models_{\mathcal{PL}} \varphi$. In particular the equivalence $\{s\} \models \varphi \Leftrightarrow s \models_{\mathcal{PL}} \varphi$ holds for every assignment s.*

The syntax of *propositional logic with intuitionistic disjunction* $\mathcal{PL}(\mathbin{\mathbb{V}})(\Phi)$ is obtained by extending the syntax of $\mathcal{PL}(\Phi)$ by the grammar rule $\varphi ::= (\varphi \mathbin{\mathbb{V}} \varphi)$. The syntax of *propositional dependence logic* $\mathcal{PD}(\Phi)$ is obtained by extending the syntax of $\mathcal{PL}(\Phi)$ by the grammar rules $\varphi ::= \mathrm{dep}(p_1, \ldots, p_n, q)$, where $p_1, \ldots, p_n, q \in \Phi$ and $n \in \mathbb{N}$. The intuitive meaning of the *propositional dependence atom* $\mathrm{dep}(p_1, \ldots, p_n, q)$ is that the truth value of the proposition symbol q is completely determined by the truth values of the proposition symbols $p_1, \ldots, p_n$. We define the semantics for the intuitionistic disjunction and the propositional dependence atoms as follows:

$$\begin{aligned}
X \models (\varphi \mathbin{\mathbb{V}} \psi) &\quad\Leftrightarrow\quad X \models \varphi \text{ or } X \models \psi\\
X \models \mathrm{dep}(p_1, \ldots, p_n, q) &\quad\Leftrightarrow\quad \forall s, t \in X : s(p_1) = t(p_1), \ldots, s(p_n) = t(p_n)\\
&\qquad\qquad \text{implies that } s(q) = t(q).
\end{aligned}$$

The next proposition is very useful. The proof is very easy and analogous to the corresponding proof for first-order dependence logic [9].

▶ **Proposition 3** (Downwards closure). *Let φ be a formula of $\mathcal{PL}(\mathbin{\mathbb{V}})$ or $\mathcal{PD}$ and let $Y \subseteq X$ be propositional teams. Then $X \models \varphi$ implies $Y \models \varphi$.*

Note that, by downwards closure, $X \models (\varphi \vee \psi)$ iff $Y \models \varphi$ and $X \setminus Y \models \psi$ for some $Y \subseteq X$.

2.2 Modal logics

In order to keep the notation light, we restrict our attention to mono-modal logic, i.e., to modal logic with just the modal operators $\Diamond$ and $\Box$. However this is not really a restriction,

since the definitions, results, and proofs of this article generalize, in a straightforward manner, to facilitate any number of modalities.

Let Φ be a set of atomic propositions. The set of formulae for *modal logic* $\mathcal{ML}(\Phi)$ is generated by the grammar: $\varphi ::= p \mid \neg p \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi) \mid \Diamond\varphi \mid \Box\varphi$, where $p \in \Phi$.

Note that, since negations are allowed only in front of proposition symbols, $\Box$ and $\Diamond$ are *not* interdefinable. The syntax of *modal logic with intuitionistic disjunction* $\mathcal{ML}(\otimes)(\Phi)$ is obtained by extending the syntax of $\mathcal{ML}(\Phi)$ by the grammar rule $\varphi ::= (\varphi \otimes \varphi)$.

The *team semantics for modal logic* is defined via *Kripke models* and *teams*. In the context of modal logic, teams are subsets of the domain of the model.

▶ **Definition 4.** Let Φ be a set of proposition symbols. A *Kripke model* K over Φ is a tuple $\mathrm{K} = (W, R, V)$, where W is a nonempty set of *worlds*, $R \subseteq W \times W$ is a binary relation, and $V : \Phi \to \mathcal{P}(W)$ is a *valuation*. A subset T of W is called a *team* of K. Furthermore define

$$R[T] := \{w \in W \mid \exists v \in T \text{ s.t. } vRw\}, \quad R^{-1}[T] := \{w \in W \mid \exists v \in T \text{ s.t. } wRv\}.$$

For teams $T, S \subseteq W$, we write $T[R]S$ if $S \subseteq R[T]$ and $T \subseteq R^{-1}[S]$. Thus, $T[R]S$ holds if and only if for every $w \in T$ there exists some $v \in S$ such that wRv, and for every $v \in S$ there exists some $w \in T$ such that wRv.

We are now ready to define the team semantics for modal logic and modal logic with intuitionistic disjunction.

▶ **Definition 5.** Let Φ be a set of atomic propositions, K a Kripke model and T a team of K. The satisfaction relation $\mathrm{K}, T \models \varphi$ for $\mathcal{ML}(\Phi)$ is defined as follows.

$$\begin{aligned}
\mathrm{K}, T \models p &\;\Leftrightarrow\; w \in V(p) \text{ for every } w \in T.\\
\mathrm{K}, T \models \neg p &\;\Leftrightarrow\; w \notin V(p) \text{ for every } w \in T.\\
\mathrm{K}, T \models (\varphi \wedge \psi) &\;\Leftrightarrow\; \mathrm{K}, T \models \varphi \text{ and } K, T \models \psi.\\
\mathrm{K}, T \models (\varphi \vee \psi) &\;\Leftrightarrow\; \mathrm{K}, T_1 \models \varphi \text{ and } \mathrm{K}, T_2 \models \psi \text{ for some } T_1 \text{ and } T_2\\
&\qquad\quad \text{such that } T_1 \cup T_2 = T.\\
\mathrm{K}, T \models \Diamond\varphi &\;\Leftrightarrow\; \mathrm{K}, T' \models \varphi \text{ for some } T' \text{ such that } T[R]T'.\\
\mathrm{K}, T \models \Box\varphi &\;\Leftrightarrow\; \mathrm{K}, T' \models \varphi, \text{ where } T' = R[T].
\end{aligned}$$

For $\mathcal{ML}(\otimes)$ we have the following additional clause:

$$\mathrm{K}, T \models (\varphi \otimes \psi) \;\Leftrightarrow\; \mathrm{K}, T \models \varphi \text{ or } \mathrm{K}, T \models \psi.$$

By $\models_{\mathcal{ML}}$, we denote the ordinary satisfaction relation of modal logic defined via pointed Kripke models.

▶ **Proposition 6** ([8]). *Let φ be an $\mathcal{ML}$-formula, K be a Kripke model, and T be a team of* K. *Then* $\mathrm{K}, T \models \varphi \Leftrightarrow \forall w \in T : \mathrm{K}, w \models_{\mathcal{ML}} \varphi$ *In particular, for every point w of* K, *the equivalence* $\mathrm{K}, \{w\} \models \varphi \Leftrightarrow \mathrm{K}, w \models_{\mathcal{ML}} \varphi$ *holds.*

The syntax for *modal dependence logic* $\mathcal{MDL}(\Phi)$ is obtained by extending the syntax of $\mathcal{ML}(\Phi)$ by the rules $\varphi ::= \mathrm{dep}(p_1, \ldots, p_n, q)$, where $p_1, \ldots, p_n, q \in \Phi$ and $n \in \mathbb{N}$, for propositional dependence atoms. The syntax for *extended modal dependence logic* $\mathcal{EMDL}(\Phi)$ is obtained by extending the syntax of $\mathcal{ML}(\Phi)$ by the rules $\varphi ::= \mathrm{dep}(\varphi_1, \ldots, \varphi_n, \psi)$, where $\varphi_1, \ldots, \varphi_n, \psi \in \mathcal{ML}(\Phi)$ and $n \in \mathbb{N}$, for *modal dependence atoms*. The intuitive meaning of

the modal dependence atom $\mathrm{dep}(\varphi_1,\dots,\varphi_n,\psi)$ is that the truth value of the formula ψ is completely determined by the truth values of the formulae $\varphi_1,\dots,\varphi_n$. Formally:

$$\mathrm{K},T \models \mathrm{dep}(\varphi_1,\dots,\varphi_n,\psi) \quad \Leftrightarrow \quad \forall w,v \in T : \bigwedge_{1\le i\le n} (\mathrm{K},\{w\} \models \varphi_i \Leftrightarrow \mathrm{K},\{v\} \models \varphi_i)$$
$$\text{implies } (\mathrm{K},\{w\} \models \psi \Leftrightarrow \mathrm{K},\{v\} \models \psi).$$

The following result for $\mathcal{MDL}$ and $\mathcal{ML}(\mathbin{⍚})$ is due to [10] and [3], respectively. For $\mathcal{EMDL}$ it follows via a translation from $\mathcal{EMDL}$ into $\mathcal{ML}(\mathbin{⍚})$, see [1].

▶ **Proposition 7** (Downwards closure). *Let φ be a formula of $\mathcal{ML}(\mathbin{⍚})$ or $\mathcal{EMDL}$, let* K *be a Kripke model and let $S \subseteq T$ be teams of* K. *Then* $\mathrm{K},T \models \varphi$ *implies* $\mathrm{K},S \models \varphi$.

2.3 Equivalence and validity in team semantics

We say that formulae φ and ψ of $\mathcal{PL}(\mathbin{⍚})(\Phi)$ or $\mathcal{PD}(\Phi)$ are *equivalent* and write $\varphi \equiv \psi$, if the equivalence $X \models \varphi \Leftrightarrow X \models \psi$ holds for every propositional team X. Likewise, we say that formulae φ and ψ of $\mathcal{ML}(\mathbin{⍚})(\Phi)$ or $\mathcal{EMDL}(\Phi)$ are *equivalent* and write $\varphi \equiv \psi$, if the equivalence $K,T \models \varphi \Leftrightarrow K,T \models \psi$ holds for every Kripke model K and team T of K.

A formula φ of $\mathcal{PL}(\mathbin{⍚})(\Phi)$ or $\mathcal{PD}(\Phi)$ is said to be *valid*, if $X \models \varphi$ holds for every team X such that the proposition symbols that occur in φ are in the domain of X. Analogously, a formula ψ of $\mathcal{EMDL}(\Phi)$ or $\mathcal{ML}(\mathbin{⍚})(\Phi)$ is said to be *valid*, if $\mathrm{K},T \models \varphi$ holds for every Kripke model K (such that the proposition symbols in φ are mapped by the valuation of K) and every team T of K. When φ is a valid formula of $\mathcal{L}$, we write $\models_{\mathcal{L}} \varphi$.

The following proposition shown in [11, 12] will later prove to be very useful.

▶ **Proposition 8** ($\mathbin{⍚}$-disjunction property). *Let $\mathcal{L} \in \{\mathcal{PL}(\mathbin{⍚}), \mathcal{ML}(\mathbin{⍚})\}$. For every φ,ψ in $\mathcal{L}$,* $\models_{\mathcal{L}} (\varphi \mathbin{⍚} \psi)$ *iff* $\models_{\mathcal{L}} \varphi$ *or* $\models_{\mathcal{L}} \psi$.

3 Extending axiomatizations of $\mathcal{PL}$ and $\mathcal{ML}$

In this section we show how to extend sound and complete axiomatizations for $\mathcal{PL}$ and $\mathcal{ML}$ into sound and complete axiomatizations for $\mathcal{PL}(\mathbin{⍚})$ and $\mathcal{ML}(\mathbin{⍚})$, respectively. We use the fact that both $\mathcal{PL}(\mathbin{⍚})$ and $\mathcal{ML}(\mathbin{⍚})$ have the $\mathbin{⍚}$-disjunction property. In addition, we obtain axiomatizations for $\mathcal{PD}$, $\mathcal{MDL}$, and $\mathcal{EMDL}$. The axiomatizations are based on compositional translations from $\mathcal{PD}$ into $\mathcal{PL}(\mathbin{⍚})$, and from $\mathcal{MDL}$ and $\mathcal{EMDL}$ into $\mathcal{ML}(\mathbin{⍚})$.

3.1 Axiomatizations for $\mathcal{PL}(\mathbin{⍚})$ and $\mathcal{ML}(\mathbin{⍚})$

In the definition below, we treat different occurrences of the same formulae as distinct entities.

▶ **Definition 9.** Let φ be a formula of $\mathcal{PL}(\mathbin{⍚})$ or $\mathcal{ML}(\mathbin{⍚})$. Let $\mathrm{SubOcc}(\varphi)$ denote the *set of exactly all occurrences of subformulas of* φ. Define

$$\mathrm{SubOcc}_{\mathbin{⍚}}(\varphi) := \{(\psi \mathbin{⍚} \theta) \mid (\psi \mathbin{⍚} \theta) \in \mathrm{SubOcc}(\varphi)\}.$$

We call a function $f : \mathrm{SubOcc}_{\mathbin{⍚}}(\varphi) \to \mathrm{SubOcc}(\varphi)$ a *$\mathbin{⍚}$-selection function for* φ if $f\big((\psi \mathbin{⍚} \theta)\big) \in \{\psi,\theta\}$, for every $(\psi \mathbin{⍚} \theta) \in \mathrm{SubOcc}_{\mathbin{⍚}}(\varphi)$. If f is a $\mathbin{⍚}$-selection function for φ, then φ^f denotes the formula that is obtained from φ by replacing simultaneously each $(\psi \mathbin{⍚} \theta) \in \mathrm{SubOcc}_{\mathbin{⍚}}(\varphi)$ by $f(\psi \mathbin{⍚} \theta)$.

Note that if $\varphi \in \mathcal{PL}(\mathbin{⍚})(\Phi)$, $\psi \in \mathcal{ML}(\mathbin{⍚})(\Psi)$, f is a $\mathbin{⍚}$-selection function for φ, and g is a $\mathbin{⍚}$-selection function for ψ, then $\varphi^f \in \mathcal{PL}(\Phi)$ and $\psi^g \in \mathcal{ML}(\Psi)$.

▶ **Proposition 10** ([11]). *Let φ be a formula of $\mathcal{PL}(\Qoppa)$ or $\mathcal{ML}(\Qoppa)$, and let F be the set of exactly all $\Qoppa$-selection functions for φ. Then, $\varphi \equiv \bigQoppa_{f\in F} \varphi^f$.*

Let $\mathbf{H}_{\mathcal{PL}}$ and $\mathbf{H}_{\mathcal{ML}}$ denote sound and complete axiomatizations of the negation normal form fragments of $\mathcal{PL}$ and $\mathcal{ML}$, respectively. For a logic $\mathcal{L}$, an *$\mathcal{L}$-context* is a formula of the logic $\mathcal{L}$ extended with the grammar rule $\varphi ::= *$. By $\varphi(\psi\,/\,*)$ we denote the formula that is obtained form φ by uniformly substituting each occurrence of $*$ in φ by ψ. We are now ready to define the axiomatizations for $\mathcal{PL}(\Qoppa)$ and $\mathcal{ML}(\Qoppa)$. We use $\mathcal{PL}(\Qoppa)$- and $\mathcal{ML}(\Qoppa)$-contexts in the following rules:

$$\frac{\varphi(\psi_i\,/\,*)}{\varphi\big((\psi_1 \Qoppa \psi_2)\,/\,*\big)}\,(I\Qoppa i) \qquad i \in \{1,2\}.$$

Let $\mathbf{H}_{\mathcal{PL}(\Qoppa)}$ ($\mathbf{H}_{\mathcal{ML}(\Qoppa)}$, resp.) be the calculus $\mathbf{H}_{\mathcal{PL}}$ ($\mathbf{H}_{\mathcal{ML}}$, resp.) extended with the rules $(I\Qoppa 1)$ and $(I\Qoppa 2)$. In the calculi $\mathbf{H}_{\mathcal{PL}(\Qoppa)}$ and $\mathbf{H}_{\mathcal{ML}(\Qoppa)}$, the axioms and inference rules of $\mathbf{H}_{\mathcal{PL}}$ and $\mathbf{H}_{\mathcal{ML}}$ may only be applied to formulae of $\mathcal{PL}$ and $\mathcal{ML}$ (i.e, to formulae without $\Qoppa$), respectively.

▶ **Theorem 11.** *$\mathbf{H}_{\mathcal{PL}(\Qoppa)}$ and $\mathbf{H}_{\mathcal{ML}(\Qoppa)}$ are sound and complete.*

Proof. We will proof the soundness and completeness for $\mathbf{H}_{\mathcal{PL}(\Qoppa)}$. The case for $\mathbf{H}_{\mathcal{ML}(\Qoppa)}$ is completely analogous. Note first that from Proposition 2 it follows directly that $\mathbf{H}_{\mathcal{PL}}$ is complete for $\mathcal{PL}$ also in the context of team semantics.

For soundness, it suffices to show that the rule $(I\Qoppa 1)$ preserves validity. The case for $(I\Qoppa 2)$ is symmetric. Let φ be a $\mathcal{PL}(\Qoppa)$-context and let ψ_1 and ψ_2 be $\mathcal{PL}(\Qoppa)$-formulae. Assume that $\gamma_1 := \varphi(\psi_1\,/\,*)$ is valid. We will show that then $\gamma_2 := \varphi\big((\psi_1 \Qoppa \psi_2)\,/\,*\big)$ is valid. Let F and G be the sets of exactly all $\Qoppa$-selection functions for γ_1 and γ_2, respectively. By Proposition 10, $\gamma_1 \equiv \bigQoppa_{f\in F}\gamma_1^f$ and $\gamma_2 \equiv \bigQoppa_{g\in G}\gamma_2^g$. Since γ_1 is valid, it follows by Proposition 8, that $\gamma_1^{f'}$ is valid for some $f' \in F$. Since clearly, for every $f \in F$, there exists some $g \in G$ such that $\gamma_1^f = \gamma_2^g$, it follows that there exists some $g' \in G$ such that $\gamma_2^{g'}$ is valid. Thus γ_2 is valid.

In order to prove completeness, assume that a $\mathcal{PL}(\Qoppa)$-formula φ is valid. Let F be the set of exactly all $\Qoppa$-selection functions for φ. By Propositions 10 and 8, there exists a function $f \in F$ such that the $\mathcal{PL}$-formula φ^f is valid. Since $\mathbf{H}_{\mathcal{PL}}$ is complete and $\mathbf{H}_{\mathcal{PL}(\Qoppa)}$ extends $\mathbf{H}_{\mathcal{PL}}$, φ^f is provable also in $\mathbf{H}_{\mathcal{PL}(\Qoppa)}$. Clearly by using the rules $(I\Qoppa 1)$ and $(I\Qoppa 2)$ repetitively to φ^f, we eventually obtain φ. Thus we conclude that $\mathbf{H}_{\mathcal{PL}(\Qoppa)}$ is complete. ◀

3.2 Axiomatizations for $\mathcal{PD}$, $\mathcal{MDL}$, and $\mathcal{EMDL}$

The following equivalence was observed by Väänänen in [10]:

$$\mathrm{dep}(p_1,\ldots,p_n,q) \equiv \bigvee_{a_1,\ldots,a_n\in\{\bot,\top\}} \bigwedge \{p_1^{a_1},\ldots,p_n^{a_n},(q \Qoppa q^{\bot})\}. \tag{1}$$

Ebbing et al. ([1]) extended this observation of Väänänen into the following equivalence concerning $\mathcal{EMDL}$:

$$\mathrm{dep}(\varphi_1,\ldots,\varphi_n,\psi) \equiv \bigvee_{a_1,\ldots,a_n\in\{\bot,\top\}} \bigwedge \{\varphi_1^{a_1},\ldots,\varphi_n^{a_n},(\psi \Qoppa \psi^{\bot})\}. \tag{2}$$

These equivalences demonstrate the existence of compositional translations from $\mathcal{PD}$ into $\mathcal{PL}(\Qoppa)$, and from $\mathcal{MDL}$ and $\mathcal{EMDL}$ into $\mathcal{ML}(\Qoppa)$, respectively.

We will use the insight that rises from combining the above equivalences with Propositions 8 and 10 in order to construct axiomatizations for $\mathcal{PD}$, $\mathcal{MDL}$, and $\mathcal{EMDL}$, respectively. Recall that when $\vec{a}$ is a finite tuple of symbols, we use a_j to denote the jth member of $\vec{a}$. For each natural number $n \in \mathbb{N}$ and function $f : \{\bot, \top\}^n \to \{\top, \bot\}$, we have the following rules:

$$\frac{\varphi\Big(\bigvee_{\vec{a}\in\{\bot,\top\}^n} \bigwedge \{p_1^{a_1}, \ldots, p_n^{a_n}, q^{f(\vec{a})}\} \,/\, *\Big)}{\varphi\big(\mathrm{dep}(p_1, \ldots, p_n, q) \,/\, *\big)} \,\big(\mathcal{PL}\,\mathrm{dep}(f)\big)$$

$$\frac{\varphi\Big(\bigvee_{\vec{a}\in\{\bot,\top\}^n} \bigwedge \{\varphi_1^{a_1}, \ldots, \varphi_n^{a_n}, \psi^{f(\vec{a})}\} \,/\, *\Big)}{\varphi\big(\mathrm{dep}(\varphi_1, \ldots, \varphi_n, \psi) \,/\, *\big)} \,\big(\mathcal{ML}\,\mathrm{dep}(f)\big)\dagger$$

where † means that $\varphi_1, \ldots, \varphi_n, \psi$ are required to be modal formulae.[1] Define $\mathcal{PL}\mathsf{dep} := \{\big(\mathcal{PL}\,\mathrm{dep}(f)\big) \mid f : \{\bot, \top\}^n \to \{\top, \bot\}$, where $n \in \mathbb{N}\}$ and $\mathcal{ML}\mathsf{dep} := \{\big(\mathcal{ML}\,\mathrm{dep}(f)\big) \mid f : \{\bot, \top\}^n \to \{\top, \bot\}$, where $n \in \mathbb{N}\}$. Let $\mathbf{H}_{\mathcal{PD}}$ and $\mathbf{H}_{\mathcal{MDL}}$ be the extensions of the calculi $\mathbf{H}_{\mathcal{PL}}$ and $\mathbf{H}_{\mathcal{ML}}$ by the rules of $\mathcal{PL}\mathsf{dep}$, respectively. Let $\mathbf{H}_{\mathcal{EMDL}}$ be the extension of $\mathbf{H}_{\mathcal{ML}}$ by the rules of $\mathcal{ML}\mathsf{dep}$.

The proof of the following theorem is analogous to that of Theorem 11.

▶ **Theorem 12.** *Let $\mathcal{L} \in \{\mathcal{PD}, \mathcal{MDL}, \mathcal{EMDL}\}$, $\mathbf{H}_{\mathcal{L}}$ is sound and complete.*

4 Labeled tableaus for propositional dependence logics

The calculi presented in Section 3 have a few clear shortcomings. Foremost, the calculi miss the team semantic nature of these logics. Thus the calculi are in some parts quite complicated. Especially this is the case for the rules $\mathcal{PL}\,\mathsf{dep}$ and $\mathcal{ML}\,\mathsf{dep}$. This seems to be the case also for any concrete implementations of the axiomatizations $\mathbf{H}_{\mathcal{PL}}$ and $\mathbf{H}_{\mathcal{ML}}$ of the negation normal form fragments of $\mathcal{PL}$ and $\mathcal{ML}$, respectively.

In this section we give axiomatizations for $\mathcal{PD}$, $\mathcal{MDL}$, and $\mathcal{EMDL}$ that do not have the shortcomings of the calculi of Section 3. The proof rules of the labeled tableau calculi that are given in this section have a natural and simple correspondence with the truth definitions of connectives and modalities in team semantics.

4.1 Checking validity via small teams

The following result (observed, e.g., in [11]) follows directly from the fact that $\mathcal{PL}(\mathbb{0})$ and $\mathcal{PD}$ are downwards closed, i.e., from Proposition 3.

▶ **Proposition 13.** *Let φ be a formula of $\mathcal{PL}(\mathbb{0})$ or $\mathcal{PD}$ and let D be the set of exactly all proposition symbols that occur in φ. Then φ is valid iff $\{0,1\}^D \models \varphi$.*

Adapting a notion that was introduced by Jarmo Kontinen in [5] for first-order dependence logic, we say that an $\mathcal{ML}(\mathbb{0})$- or $\mathcal{EMDL}$-formula φ is *n-coherent* if the condition

$$\mathrm{K}, T \models \varphi \quad \Leftrightarrow \quad \mathrm{K}, T' \models \varphi \text{ for all } T' \subseteq T \text{ such that } |T'| \leq n$$

holds for all Kripke models K and teams T of K.

[1] In the special case where φ is $*$ in the rule $\big(\mathcal{PL}\,\mathrm{dep}(f)\big)$, the obtained rule coincides with the rule of Dependence Atom Introduction in [12, p.75].

The following result for $\mathcal{ML}(\oslash)$ was shown in [4]. The result for $\mathcal{EMDL}$ follows from the result for $\mathcal{ML}(\oslash)$ essentially via the following equivalence.

$$\mathrm{dep}(\varphi_1,\dots,\varphi_n,\psi) \equiv \bigvee_{a_1,\dots,a_n\in\{\bot,\top\}} \bigwedge \{\varphi_1^{a_1},\dots,\varphi_n^{a_n},(\psi \oslash \psi^{\bot})\}.$$

For $\varphi \in \mathcal{ML}(\oslash)$, we define $\mathrm{Rank}_{\oslash}(\varphi)$ to be the number of intuitionistic disjunctions in φ. For $\psi \in \mathcal{EMDL}$, we define $\mathrm{Rank}_{\oslash}(\psi)$ to be the number of intuitionistic disjunctions in the $\mathcal{ML}(\oslash)$ formula obtained by using the above equivalence. Note that $\mathrm{Rank}_{\oslash}(\varphi) \leq |\varphi|$, whereas $\mathrm{Rank}_{\oslash}(\psi) \leq 2^{|\psi|}$.

▶ **Theorem 14.** *Every formula φ of $\mathcal{ML}(\oslash)$ or $\mathcal{EMDL}$ is $2^{\mathrm{Rank}_{\oslash}(\varphi)}$-coherent.*

The following result follows directly from Theorem 14.

▶ **Corollary 15.** *Let φ be a formula of $\mathcal{ML}(\oslash)$ or $\mathcal{EMDL}$. The following holds:*

φ is valid iff $\mathrm{K},T \models \varphi$ for every Kripke model K and every team T of K such that $|T| \leq 2^{\mathrm{Rank}_{\oslash}(\varphi)}$.

4.2 Tableau Calculi for $\mathcal{PL}$, $\mathcal{PL}(\oslash)$, and $\mathcal{PD}$

We will now present labeled tableau calculi for $\mathcal{PL}$, $\mathcal{PL}(\oslash)$, and $\mathcal{PD}$. In Section 4.3 we will extend these calculi to deal with $\mathcal{ML}$, $\mathcal{MDL}$, and $\mathcal{EMDL}$.

Any finite, possibly empty, subset $\alpha \subseteq \mathbb{N}$ is called a *label*. We mainly use symbols $\alpha,\beta,\alpha_1,\alpha_2,\beta_1,\beta_2$, etc, in order to refer to labels and symbols i,j,i_1,i_2,j_1,j_2, etc, in order to refer to natural numbers. Our tableau calculi are labeled, meaning that the formulae occurring in the tableau rules are *labeled formulae*, i.e., of the form $\alpha : \varphi$, where α a label and φ is a formula of some logic $\mathcal{L}$. Labels correspond to teams and the elements of labels, i.e., natural numbers, correspond to points in a model. The intended *top down* reading of the labeled formula $\alpha : \varphi$ is that α denotes some team that *falsifies* φ. A tableau in these calculi is just a well-founded, finitely branching tree in which each node is labeled by a labeled formula, and the edges represent applications of the tableau rules. The tableau rules needed for axiomatizing $\mathcal{PL}$, $\mathcal{PL}(\oslash)$, and $\mathcal{PD}$ are given in Figure 1.

In the construction of tableaus, we impose a rule that a labeled formula is never added to a tableau branch in which it already occurs. A *saturated branch* is a tableau branch in which no rules can be applied or the application of the rules have no effect on the branch. A *saturated tableau* is a tableau in which every branch is saturated. A branch of a tableau is called *closed* if it contains at least one of the following:

1. Both $\{i\} : p$ and $\{i\} : \neg p$, for some proposition symbol p and natural number $i \in \mathbb{N}$.
2. $\emptyset : \varphi$, for some formula φ.
3. $\{i\} : \mathrm{dep}(p_1,\dots,p_n,q)$, for some proposition symbols $p_1,\dots,p_n,q$ and $i,n \in \mathbb{N}$.

If a branch of a tableau is not closed it is called *open*. A tableau is called *closed* if every branch of the tableau is closed. A tableau is called *open* if at least one branch in the tableau is open.

Let $\mathbf{T}_{\mathcal{PL}}$ denote the calculi consisting of the rules $(Prop)$, $(\neg Prop)$, $(\wedge)$, and $(\vee)$ of Figure 1. Let $\mathbf{T}_{\mathcal{PL}(\oslash)}$ denote the extension of $\mathbf{T}_{\mathcal{PL}}$ by the rule $(\oslash)$ of Figure 1, and $\mathbf{T}_{\mathcal{PD}}$ denote the extension of $\mathbf{T}_{\mathcal{PL}}$ by the rules $(Split)$ and $(\mathcal{PL}\,dep)$ of Figure 1.

Let φ be a formula of $\mathcal{L}(\Phi) \in \{\mathcal{PL}(\Phi),\mathcal{PL}(\oslash)(\Phi),\mathcal{PD}(\Phi)\}$ and $k := \min(|\Phi|,\mathrm{Rank}_{\oslash}(\varphi))$. We say that a tableau $\mathcal{T}$ is a *tableau for* φ if the root of $\mathcal{T}$ is $\{1,\dots,2^k\} : \varphi$ and $\mathcal{T}$ is obtained

$$\frac{\{i_1,\dots,i_k\} : p}{\{i_1\} : p \mid \dots \mid \{i_k\} : p}\,(Prop) \qquad \frac{\{i_1,\dots,i_k\} : \neg p}{\{i_1\} : \neg p \mid \dots \mid \{i_k\} : \neg p}\,(\neg Prop) \qquad \frac{\alpha : (\varphi \wedge \psi)}{\alpha : \varphi \mid \alpha : \psi}\,(\wedge)$$

$$\frac{\alpha : (\varphi \vee \psi)}{\beta : \varphi \mid \alpha \setminus \beta : \psi}\,(\vee) \text{ where } \beta \subseteq \alpha \qquad \frac{\alpha : (\varphi \mathbin{\mathbb{Y}} \psi)}{\begin{array}{c}\alpha : \varphi \\ \alpha : \psi\end{array}}\,(\mathbb{Y})$$

$$\frac{\alpha : \mathrm{dep}(p_1,\dots,p_n,q)}{\alpha_1 : \mathrm{dep}(p_1,\dots,p_n,q) \mid \dots \mid \alpha_k : \mathrm{dep}(p_1,\dots,p_n,q)}\,(Split)\dagger$$

†: $\alpha_1,\dots,\alpha_k$ are exactly all subsets of α of cardinality 2.

$$\frac{\{i_1,i_2\} : \mathrm{dep}(p_1,\dots,p_n,q)}{\begin{array}{c}\{i_1\} : p_1^{g_1(1)} \mid \dots \mid \{i_1\} : p_1^{g_k(1)} \\ \{i_2\} : p_1^{g_1(1)} \mid \dots \mid \{i_2\} : p_1^{g_k(1)} \\ \vdots \qquad\qquad \vdots \qquad\qquad \vdots \\ \{i_1\} : p_n^{g_1(n)} \mid \dots \mid \{i_1\} : p_n^{g_k(n)} \\ \{i_2\} : p_n^{g_1(n)} \mid \dots \mid \{i_2\} : p_n^{g_k(n)} \\ \{i_1,i_2\} : q \mid \dots \mid \{i_1,i_2\} : q \\ \{i_1,i_2\} : \neg q \mid \dots \mid \{i_1,i_2\} : \neg q\end{array}}\,(\mathcal{PL}\,dep)\ddagger$$

‡: $g_1,\dots g_k$ are exactly all functions with domain $\{1,\dots,n\}$ and co-domain $\{\top,\bot\}$.

Figure 1 Tableau Rules for $\mathbf{T}_{\mathcal{PL}}$, $\mathbf{T}_{\mathcal{PL}(\mathbb{Y})}$, and $\mathbf{T}_{\mathcal{PD}}$.

by applying the rules of $\mathbf{T}_{\mathcal{L}}$. We say that φ is *provable* in $\mathbf{T}_{\mathcal{L}}$ and write $\vdash_{\mathbf{T}_{\mathcal{L}}} \varphi$ if there exists a closed tableau for φ.

▶ **Example 16.** We show that the $\mathcal{PD}$-formula $\mathrm{dep}(p,p)$ is provable $\mathbf{T}_{\mathcal{PD}}$. Figure 2 is an illustration of a closed $\mathbf{T}_{\mathcal{PD}}$-tableau for $\mathrm{dep}(p,p)$.

Since the number of proposition symbols that occur in $\mathrm{dep}(p,p)$ is one, the root of the tableau is $\{1,2\} : \mathrm{dep}(p,p)$. We first apply the rule $(\mathcal{PL}dep)$ to $\{1,2\} : \mathrm{dep}(p,p)$ and branch into two branches as depicted in Figure 2. In the left (right) branch we apply the rule $(\neg Prop)$ to $\{1,2\} : \neg p$ ($(Prop)$ to $\{1,2\} : p$). Consequently, each branch of the tableau becomes closed due to the labeled formulae of the type $\{i\} : p$ and $\{i\} : \neg p$, $i \in \{1,2\}$. Therefore, $\mathrm{dep}(p,p)$ is a theorem of $\mathbf{T}_{\mathcal{PD}}$.

▶ **Theorem 17** (Termination of $\mathbf{T}_{\mathcal{PL}}$, $\mathbf{T}_{\mathcal{PL}(\mathbb{Y})}$, and $\mathbf{T}_{\mathcal{PD}}$)**.** *Let $\mathcal{L}$ be a logic in $\{\mathcal{PL}, \mathcal{PL}(\mathbb{Y}), \mathcal{PD}\}$ and φ an $\mathcal{L}$-formula. Every tableau for φ in $\mathbf{T}_{\mathcal{L}}$ is finite.*

Proof. Let $\mathcal{T}$ be a tableau for φ. By definition, the root of $\mathcal{T}$ is $\alpha : \varphi$, for some finite α. Clearly every application of the tableau rules either decreases the size of the label or the length of the formula. Note also that the rule $(\vee)$ can be applied to any $\beta : \psi \in \mathcal{T}$ only finitely many times. Thus $\mathcal{T}$ must be finite. ◀

▶ **Lemma 18.** *If there exists a saturated open branch for φ then φ is not valid.*

Proof. Let $\mathcal{B}$ be a saturated open branch for φ and let Φ be the set of proposition symbols that occur in φ. Let $\alpha : \varphi$ denote the root of the branch $\mathcal{B}$. It is easy to check that if $\beta : \psi$ is a labeled formula in $\mathcal{B}$ then $\beta \subseteq \alpha$. For each $i \in \alpha$ we define an assignment $s_i : \Phi \to \{0,1\}$ such that

$$s_i(p) := \begin{cases} 1 & \text{if the labeled formula } \{i\} : \neg p \text{ occurs in the branch } \mathcal{B}, \\ 0 & \text{otherwise.} \end{cases}$$

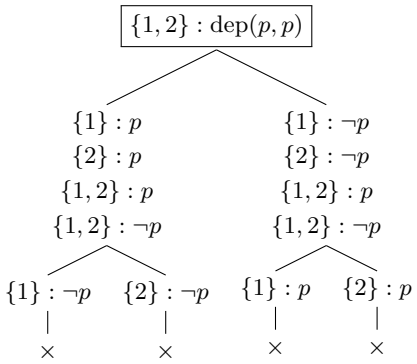


Figure 2 A tableau showing that the $\mathcal{PD}$-formula $\mathrm{dep}(p,p)$ is provable in $\mathbf{T}_{\mathcal{PD}}$.

It is easy to show by induction that if a labeled formula $\beta : \psi$ occurs in the branch $\mathcal{B}$ then $X_\beta \not\models \psi$, where $X_\beta = \{s_i \mid i \in \beta\}$. Thus φ is not valid. ◀

▶ **Theorem 19** (Completeness of $\mathbf{T}_{\mathcal{PL}}$, $\mathbf{T}_{\mathcal{PL}(\mathbb{V})}$, and $\mathbf{T}_{\mathcal{PD}}$)**.** *Let $\mathcal{L}$ be any of the logics in $\{\mathcal{PL}, \mathcal{PL}(\mathbb{V}), \mathcal{PD}\}$. The calculus $\mathbf{T}_{\mathcal{L}}$ is complete.*

Proof. Fix $\mathcal{L} \in \{\mathcal{PL}, \mathcal{PL}(\mathbb{V}), \mathcal{PD}\}$. Assume $\nvdash_{\mathbf{T}_{\mathcal{L}}} \varphi$. Thus every tableau for φ is open. From Theorem 17 it follows that there exists a saturated open tableau for φ. Thus there exists a saturated open branch for φ. Thus, by Lemma 18, $\not\models_{\mathcal{L}} \varphi$. ◀

▶ **Definition 20.** Let $\mathcal{B}$ be a tableau branch and $\mathrm{Index}(\mathcal{B})$ the set of exactly all natural numbers that occur in $\mathcal{B}$. We say that $\mathcal{B}$ is *faithful* to a propositional team X by a mapping $f : \mathrm{Index}(\mathcal{B}) \to X$ if, for all $\alpha : \varphi \in \mathcal{B}$, $f[\alpha] \not\models \varphi$.

▶ **Lemma 21.** *Let $\mathcal{L}$ be a logic in $\{\mathcal{PL}, \mathcal{PL}(\mathbb{V}), \mathcal{PD}\}$. If $\varphi \in \mathcal{L}$ is not valid then there is an open saturated branch in every saturated tableau of φ in $\mathbf{T}_{\mathcal{L}}$.*

Proof. Assume $\not\models_{\mathcal{L}} \varphi$. Let Φ be the set of exactly all proposition symbols that occur in φ. By Proposition 13, $\{0,1\}^\Phi \not\models \varphi$. Put $\alpha := \{1, \ldots, 2^{|\Phi|}\}$ and fix a bijection $f : \alpha \to \{0,1\}^\Phi$. Let $\mathcal{T}$ be an arbitrary saturated tableau for φ. By Theorem 17, $\mathcal{T}$ is finite and, by definition, the root of $\mathcal{T}$ is $\alpha : \varphi$. Note that $\mathrm{Index}(\mathcal{B}) = \alpha$, for every branch $\mathcal{B}$ with the root $\alpha : \varphi$. We will show that there is an open saturated branch in $\mathcal{T}$.

First, we establish that $\mathcal{B}_0 := \{\alpha : \varphi\}$ is faithful to $\{0,1\}^\Phi$ by f. But, this is easy since $f[\alpha] = \{0,1\}^\Phi$. Second, assume that we have constructed a branch $\mathcal{B}_n$ such that $\mathcal{B}_n$ is faithful to $\{0,1\}^\Phi$ by f. We will show that at least one extension of $\mathcal{B}_n$ by rules of $\mathbf{T}_{\mathcal{L}}$ is faithful to $\{0,1\}^\Phi$ by f. Here we are concerned with the rule of $(\vee)$ alone. Assume that, from $\beta_1 : (\psi_1 \vee \psi_2) \in \mathcal{B}_n$ and the rule of $(\vee)$, we obtain two extensions $\{\beta_2 : \psi_1\} \cup \mathcal{B}_n$ and $\{\beta_1 \setminus \beta_2 : \psi_2\} \cup \mathcal{B}_n$ for $\beta_2 \subseteq \beta_1$. Our goal is to show that one of the extensions is faithful to $\{0,1\}^\Phi$ by f. By assumption, we obtain $f[\beta_1] \not\models (\psi_1 \vee \psi_2)$. By the semantic clause for $\vee$, $f[\beta_2] \not\models \psi_1$ or $f[\beta_1] \setminus f[\beta_2] \not\models \psi_2$. Since $f[\beta_1] \setminus f[\beta_2] \subseteq f[\beta_1 \setminus \beta_2]$, it follows from downwards closure that $f[\beta_2] \not\models \psi_1$ or $f[\beta_1 \setminus \beta_2] \not\models \psi_2$. This implies that at least one of the two extensions is faithful to $\{0,1\}^\Phi$ by f. We choose one of the faithful extensions as $\mathcal{B}_{n+1}$.

Since $\mathcal{T}$ is finite and saturated, $\mathcal{B}_j$ is a saturated branch in $\mathcal{T}$ for some $j \in \mathbb{N}$. Moreover, since $\mathcal{B}_j$ is faithful to $\{0,1\}^\Phi$ by f, $\mathcal{B}_j$ is open. ◀

▶ **Theorem 22** (Soundness of $\mathbf{T}_{\mathcal{PL}}$, $\mathbf{T}_{\mathcal{PL}(\mathbb{V})}$, and $\mathbf{T}_{\mathcal{PD}}$)**.** *Let $\mathcal{L}$ be any of the logics in $\{\mathcal{PL}, \mathcal{PL}(\mathbb{V}), \mathcal{PD}\}$. The calculus $\mathbf{T}_{\mathcal{L}}$ is sound.*

$$\frac{\begin{array}{c} i_1 \mathsf{R} j_1 \\ \vdots \\ i_n \mathsf{R} j_n \\ \{i_1,\dots,i_n\} : \Diamond\varphi \end{array}}{\{j_1,\dots,j_n\} : \varphi}\ (\Diamond) \qquad \frac{\alpha : \Box\varphi}{\begin{array}{c} f_1(1)\mathsf{R}i_1 \mid \dots \mid f_k(1)\mathsf{R}i_1 \\ \vdots \quad\quad \vdots \quad\quad \vdots \\ f_1(t)\mathsf{R}i_t \mid \dots \mid f_k(t)\mathsf{R}i_t \\ \{i_1,\dots i_t\} : \varphi \mid \dots \mid \{i_1,\dots i_t\} : \varphi \end{array}}\ (\Box)\dagger$$

$$\frac{\{i_1,i_2\} : \mathrm{dep}(\varphi_1,\dots,\varphi_n,\psi)}{\begin{array}{c} \{i_1\} : \varphi_1^{h_1(1)} \mid \dots \mid \{i_1\} : \varphi_1^{h_k(1)} \\ \{i_2\} : \varphi_1^{h_1(1)} \mid \dots \mid \{i_2\} : \varphi_1^{h_k(1)} \\ \vdots \quad\quad\quad \vdots \quad\quad\quad \vdots \\ \{i_1\} : \varphi_n^{h_1(n)} \mid \dots \mid \{i_1\} : \varphi_n^{h_k(n)} \\ \{i_2\} : \varphi_n^{h_1(n)} \mid \dots \mid \{i_2\} : \varphi_n^{h_1(n)} \\ \{i_1,i_2\} : \psi \mid \dots \mid \{i_1,i_2\} : \psi \\ \{i_1,i_2\} : \psi^{\perp} \mid \dots \mid \{i_1,i_2\} : \psi^{\perp} \end{array}}\ (\mathcal{ML}\,dep)\ddagger$$

†: $t = 2^{\mathrm{Rank}_{\mathbb{V}}(\varphi)}$ and $f_1,\dots,f_k$ denote exactly all functions with domain $\{1,\dots,t\}$ and co-domain α, and $i_1,\dots,i_t$ are fresh and distinct.
‡: $h_1,\dots h_k$ denotes all the functions with domain $\{1,\dots,n\}$ and co-domain $\{\top,\perp\}$.

Figure 3 Additional Tableau Rules for $\mathbf{T}_{\mathcal{ML}}$, $\mathbf{T}_{\mathcal{ML}(\mathbb{V})}$, $\mathbf{T}_{\mathcal{MDL}}$ and $\mathbf{T}_{\mathcal{EMDL}}$.

Proof. Fix $\mathcal{L} \in \{\mathcal{PL}, \mathcal{PL}(\mathbb{V}), \mathcal{PD}\}$. Assume that $\not\models_{\mathcal{L}} \varphi$. By Lemma 21, there is an open saturated branch in every saturated tableau of φ in $\mathbf{T}_{\mathcal{L}}$. Therefore, and since, by Theorem 17, every tableau of φ in $\mathbf{T}_{\mathcal{L}}$ is finite, there does not exists any closed tableau for φ in $\mathbf{T}_{\mathcal{L}}$. Thus $\not\vdash_{\mathbf{T}_{\mathcal{L}}} \varphi$. ◀

4.3 Tableau Calculi for $\mathcal{ML}$, $\mathcal{ML}(\mathbb{V})$, $\mathcal{MDL}$, and $\mathcal{EMDL}$

In addition to labeled formulae, the tableau rules for modal logics contain *accessibility formulae* of the form $i\mathsf{R}j$, where $i, j \in \mathbb{N}$. The intended interpretation of $i\mathsf{R}j$ is that the point denoted by j is accessible by the relation R from the point denoted by i. The tableau rules for the calculi are given in Figures 1 and 3.

In the construction of tableaus, in addition to the rules given in Section 4.2, we impose that the tableau rule $(\Box)$ is never applied twice to the same labeled formula in any branch. The definitions of open, closed and saturated tableau and branch are as in Section 4.2 with the following additional rule: A branch is called *closed* also if it contains a labeled formula $\{i\} : \mathrm{dep}(\varphi_1,\dots,\varphi_n,\psi)$, for some $i, n \in \mathbb{N}$ and $\varphi_1,\dots,\varphi_n,\psi \in \mathcal{ML}$.

Let $\mathbf{T}_{\mathcal{ML}}$, $\mathbf{T}_{\mathcal{ML}(\mathbb{V})}$, and $\mathbf{T}_{\mathcal{MDL}}$ denote the extensions of $\mathbf{T}_{\mathcal{PL}}$, $\mathbf{T}_{\mathcal{PL}(\mathbb{V})}$, and $\mathbf{T}_{\mathcal{PD}}$ by the rules $(\Diamond)$ and $(\Box)$ of Figure 3, respectively. Let $\mathbf{T}_{\mathcal{EMDL}}$ denote the extension of $\mathbf{T}_{\mathcal{ML}}$ by the rules (*Split*) of Figure 1 and $(\mathcal{ML}\,dep)$ of Figure 3.

Let φ be a formula of $\mathcal{L} \in \{\mathcal{ML}, \mathcal{ML}(\mathbb{V}), \mathcal{MDL}, \mathcal{EMDL}\}$. We say that a tableau $\mathcal{T}$ is a *tableau for* φ if the root of $\mathcal{T}$ is $\{1,\dots,2^{\mathrm{Rank}_{\mathbb{V}}(\varphi)}\} : \varphi$ and $\mathcal{T}$ is obtained by applying the rules of $\mathbf{T}_{\mathcal{L}}$. We say that φ is *provable* in $\mathbf{T}_{\mathcal{L}}$ and write $\vdash_{\mathbf{T}_{\mathcal{L}}} \varphi$ if there exists a closed tableau for φ.

▶ **Example 23.** This example illustrates one difference between $\mathbf{T}_{\mathcal{PL}}$ and $\mathbf{T}_{\mathcal{MDL}}$ even for the same formula $\mathrm{dep}(p,p)$. Figure 4 is an illustration of a closed $\mathbf{T}_{\mathcal{MDL}}$-tableau for $\mathrm{dep}(p,p)$. When $\mathrm{dep}(p,p)$ is considered as a $\mathcal{PD}$-formula, the calculation starts with the label $\{1,2\}$

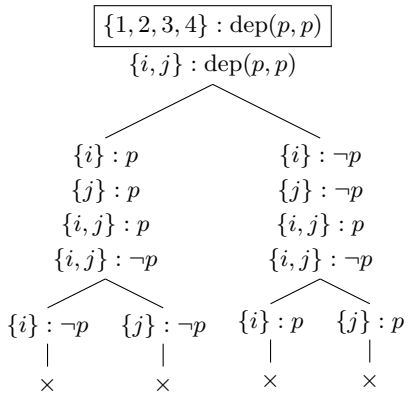


Figure 4 A tableau showing that the $\mathcal{MDL}$-formula $\mathrm{dep}(p,p)$ is provable in $\mathbf{T}_{\mathcal{MDL}}$.

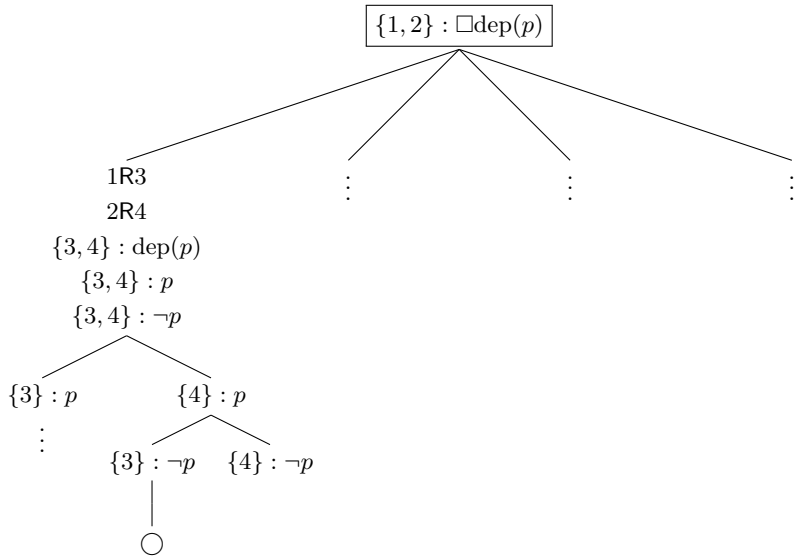


Figure 5 A tableau showing that the $\mathcal{MDL}$-formula $\Box\mathrm{dep}(p)$ is not valid.

(see Example 16 and Figure 2). However, when $\mathrm{dep}(p,p)$ is considered as an $\mathcal{MDL}$-formula, our definition leads us to start the calculation with the label $\{1,2,3,4\}$.

The equivalent $\mathcal{ML}(\mathbin{⫽})$ formula that the $\mathcal{MDL}$-formula $\mathrm{dep}(p,p)$ translates into is

$$\bigvee_{a\in\{\top,\bot\}} \bigwedge\{p^a, p \mathbin{⫽} \neg p\}.$$

Therefore $\mathrm{Rank}_{⫽}(\mathrm{dep}(p,p)) = 2$, and thus the root of any $\mathbf{T}_{\mathcal{MDL}}$-tableau for $\mathrm{dep}(p,p)$ is $\{1,2,3,4\} : \mathrm{dep}(p,p)$. We first apply the rule $(Split)$ to $\{1,2,3,4\} : \mathrm{dep}(p,p)$ and obtain 6 branches. By symmetry, we may concentrate on one of the branches. We denote it by $\{i,j\}$ $(i \neq j)$. We then apply the rule $(\mathcal{PL}dep)$ to $\{i,j\} : \mathrm{dep}(p,p)$ and branch into two branches as depicted in Figure 4. In the left (right) branch we apply the rule $(\neg Prop)$ to $\{i,j\} : \neg p$ $((Prop)$ to $\{i,j\} : p)$. Consequently, each branch of the tableau becomes closed due to the labeled formulae of the type $\{l\} : p$ and $\{l\} : \neg p$, $l \in \{i,j\}$. Therefore, $\mathrm{dep}(p,p)$ is a theorem of $\mathbf{T}_{\mathcal{MDL}}$.

▶ **Example 24.** We show that the $\mathcal{MDL}$ formula $\Box\mathrm{dep}(p)$ is not valid. Note that the equivalent $\mathcal{ML}(\mathbin{\mathbb{V}})$-formula that $\Box\mathrm{dep}(p)$ translates into is $\Box(p \mathbin{\mathbb{V}} \neg p)$. Therefore $\mathrm{Rank}_{\mathbb{V}}(\Box\mathrm{dep}(p)) = 1$, and thus the root of any $\mathbf{T}_{\mathcal{MDL}}$-tableau for $\Box\mathrm{dep}(p)$ is $\{1,2\} : \Box\mathrm{dep}(p)$. We are going to find an open saturated branch for $\Box\mathrm{dep}(p)$.

First, we apply the rule $(\Box)$ for $\{1,2\} : \Box\mathrm{dep}(p)$. One of the branches that is obtained is depicted in Figure 5. We then apply the rule $(\mathcal{PL}dep)$ to $\{3,4\} : \mathrm{dep}(p)$. Then, by applying the rules $(Prop)$ and $(\neg Prop)$ to $\{3,4\} : p$ and $\{3,4\} : \neg p$, respectively, we obtain an open saturated branch as depicted in Figure 5. From the open saturated branch, we can construct the following Kripke model $\mathrm{K} = (W, R, V)$ that falsifies the $\mathcal{MDL}$-formula $\Box\mathrm{dep}(p)$. Define $W := \{w_1, w_2, w_3, w_4\}$, $R := \{(w_1, w_3), (w_2, w_4)\}$, $V(p) := \{w_3\}$. One can easily verify that $\mathrm{K}, \{w_1, w_2\} \not\models \Box\mathrm{dep}(p)$.

▶ **Definition 25.** Let $\mathcal{L} \in \{\mathcal{ML}, \mathcal{ML}(\mathbin{\mathbb{V}}), \mathcal{MDL}, \mathcal{EMDL}\}$. Let $\mathcal{B}$ be a branch of a tableau in $\mathbf{T}_{\mathcal{L}}$ and let $\alpha : \varphi$ be the root of $\mathcal{B}$. Recall that $\mathrm{Index}(\mathcal{B})$ denotes the set of exactly all natural numbers that occur in $\mathcal{B}$. For $i, j \in \mathrm{Index}(\mathcal{B})$, we write $i \prec_{\mathcal{B}} j$ if iRj occurs in $\mathcal{B}$. By $\prec^*_{\mathcal{B}}$ and $\preceq^*_{\mathcal{B}}$, we mean the transitive closure and the reflexive and transitive closure of $\prec_{\mathcal{B}}$, respectively. Moreover, for $i \in \mathrm{Index}(\mathcal{B})$ and $n \in \mathbb{N}$, define

$$\mathrm{Level}_{\mathcal{B}}(i) := |\{j \in \mathrm{Index}(\mathcal{B}) \mid i_0 \prec^*_{\mathcal{B}} j \preceq^*_{\mathcal{B}} i, \text{ for some } i_0 \in \alpha\}|,$$
$$\mathrm{Layer}_{\mathcal{B}}(n) := \{j \in \mathrm{Index}(\mathcal{B}) \mid \mathrm{Level}_{\mathcal{B}}(j) = n\}.$$

It is easy to see that, for every branch $\mathcal{B}$, the graph $(\mathrm{Index}(\mathcal{B}), \prec_{\mathcal{B}})$ is a well-founded forest.

▶ **Theorem 26** (Termination of $\mathbf{T}_{\mathcal{ML}}$, $\mathbf{T}_{\mathcal{ML}(\mathbin{\mathbb{V}})}$, $\mathbf{T}_{\mathcal{MDL}}$, and $\mathbf{T}_{\mathcal{EMDL}}$)**.** *Let φ be a formula of $\mathcal{ML}$, $\mathcal{ML}(\mathbin{\mathbb{V}})$, $\mathcal{MDL}$, or $\mathcal{EMDL}$. Every tableau for φ is finite.*

Proof. Let $\mathcal{T}$ be a tableau for φ and let $\alpha : \varphi$ denote the root of $\mathcal{T}$. By definition α is finite. Clearly, by the definitions of the tableau rules, if $\beta : \psi$ occurs in $\mathcal{T}$ then $|\beta| \leq |\alpha|$. From this and from the definitions of the tableau rules, it is easy to see that $\mathcal{T}$ is a finitely branching tree. Thus from König's lemma it follows that $\mathcal{T}$ is infinite if and only if $\mathcal{T}$ has an infinite branch.

Let $\mathcal{B}$ be an arbitrary branch of $\mathcal{T}$. We will show that $\mathcal{B}$ is finite.

Claim 1. If $\alpha : \varphi$ occurs in $\mathcal{B}$ then, for every $i, j \in \alpha$, $\mathrm{Level}_{\mathcal{B}}(i) = \mathrm{Level}_{\mathcal{B}}(j)$.

Claim 2. For each $k \in \mathbb{N}$ the set $\mathrm{Layer}_{\mathcal{B}}(k)$ is finite.

Claim 3. There is a $k \in \mathbb{N}$ such that $\mathrm{Layer}_{\mathcal{B}}(k) = \emptyset$.

Note first that if $\mathrm{Layer}_{\mathcal{B}}(k) = \emptyset$ then $\mathrm{Layer}_{\mathcal{B}}(n) = \emptyset$, for every $n \geq k$. Thus from Claims 2 and 3 it follows that only finitely many labels may occur in $\mathcal{B}$. Note also that, for every labeled formula $\beta : \psi$ that occurs in $\mathcal{B}$, ψ is either a subformula of φ or a subformula of some $\theta^{\perp}$, where θ is an $\mathcal{ML}$ subformula of φ. Thus only finitely many labeled formulae may occur in $\mathcal{B}$. Thus $\mathcal{B}$ is finite.

Proof of Claim 1 is easy. We will sketch the proofs of Claims 2 and 3.

Proof sketch of Claim 2. Claim 2 follows from Claim 1 by induction: Clearly $\mathrm{Layer}_{\mathcal{B}}(0)$ is finite. $\mathrm{Layer}_{\mathcal{B}}(k+1)$ is generated via applications of the tableau rule $(\Box)$ to labeled formulae $\beta : \Box\psi$ of the branch $\mathcal{B}$, where $\beta \subseteq \mathrm{Layer}_{\mathcal{B}}(k)$ and $\Box\psi$ is either a subformula of φ or a subformula of some $\theta^{\perp}$, where θ is an $\mathcal{ML}$ subformula of φ. Since $\mathrm{Layer}_{\mathcal{B}}(k)$ is finite, $\mathrm{Layer}_{\mathcal{B}}(k+1)$ is as well.

Proof sketch of Claim 3. For finite labels β, define

$$m_{\mathcal{B}}(\beta) := \max\{|\varphi| \mid \beta_1 : \varphi \text{ occurs in } \mathcal{B} \text{ and } \beta_1 \cap \beta \neq \emptyset\}.$$

For finite labels β, define $M_{\mathcal{B}}(\beta : \psi) := (m_{\mathcal{B}}(\beta), |\psi|, |\beta|)$. The ordering between the tuples is defined as follows:

$$(i,j,k) < (k,l,m) \text{ iff } i < k \text{ or } (i = k \text{ and } j < l) \text{ or } (i = k \text{ and } j = l \text{ and } k < m).$$

Note that for every labeled formula $\beta : \psi$ that occurs in $\mathcal{B}$ it holds that $m_{\mathcal{B}}(\beta) < m_{\mathcal{B}}(\alpha)$, $|\psi| \leq |\varphi|$ and $|\beta| \leq |\alpha|$. Thus the ordering of the tuples is well-founded. Furthermore it is easy to check that an application of each tableau rule decreases the measure $M_{\mathcal{B}}$. For finite collections of labeled formulae Γ, define $\mathcal{M}_{\mathcal{B}}(\Gamma) := \max\{M_{\mathcal{B}}(\beta : \psi) \mid \beta : \psi \in \Gamma\}$. It is straightforward to show that, for every $k \in \mathbb{N}$, either $\mathcal{M}_{\mathcal{B}}\big(\mathrm{Layer}_{\mathcal{B}}(k+1)\big) < \mathcal{M}_{\mathcal{B}}\big(\mathrm{Layer}_{\mathcal{B}}(k)\big)$ or $\mathrm{Layer}_{\mathcal{B}}(k+1) = \emptyset$. From this the claim follows. ◀

▶ **Definition 27.** Let $\mathcal{B}$ be a tableau branch. We say that $\mathcal{B}$ is *faithful* to a Kripke model $\mathrm{K} = (W, R, V)$ if there exists a mapping $f : \mathrm{Index}(\mathcal{B}) \to W$ such that, $\mathrm{K}, f[\alpha] \not\models \varphi$ for all $\alpha : \varphi \in \mathcal{B}$, and $f(i)Rf(j)$ holds, for every $i\mathsf{R}j \in \mathcal{B}$.

▶ **Lemma 28.** *Let $\mathcal{L} \in \{\mathcal{ML}, \mathcal{ML}(\mathbb{O}), \mathcal{MDL}, \mathcal{EMDL}\}$. If $\varphi \in \mathcal{L}$ is not valid then there is an open saturated branch in every saturated tableau of φ in $\mathbf{T}_{\mathcal{L}}$.*

Proof. In this proof, we focus on $\mathcal{ML}(\mathbb{O})$. Assume that $\varphi \in \mathcal{ML}(\mathbb{O})$ is not valid. By Corollary 15, there is a Kripke model $\mathrm{K} = (W, R, V)$ and a team T of K such that $|T| \leq 2^{\mathrm{Rank}_{\mathbb{O}}(\varphi)}$ and $\mathrm{K}, T \not\models \varphi$. Put $\alpha_0 := \{1, \ldots, 2^{\mathrm{Rank}_{\mathbb{O}}(\varphi)}\}$. Let $\mathcal{T}$ be an arbitrary saturated tableau for φ. By Theorem 26, $\mathcal{T}$ is finite and, by definition, the root of $\mathcal{T}$ is $\alpha_0 : \varphi$. We will show that there is an open branch $\mathcal{B}$ in $\mathcal{T}$.

We first establish that $\mathcal{B}_0 := \{\alpha_0 : \varphi\}$ is faithful to K. Let $f : \alpha_0 \to W$ be any mapping (note: W is non-empty) such that $f[\alpha_0] = T$. Clearly $\mathrm{K}, f[\alpha_0] \not\models \varphi$, and thus $\mathcal{B}_0$ is faithful to K. Assume then that we have constructed a branch $\mathcal{B}_n$ such that $\mathcal{B}_n$ is faithful to K. Thus there is a mapping $g : \mathrm{Index}(\mathcal{B}_n) \to W$ such that, for all $\beta : \psi \in \mathcal{B}_n$, $\mathrm{K}, g[\beta] \not\models \psi$, and, for all $i\mathsf{R}j \in \mathcal{B}_n$, $g(i)Rg(j)$ holds. We will show that any rule-application to $\mathcal{B}_n$ generates at least one faithful extension $\mathcal{B}_{n+1}$ to K. Here we are concerned with the rules of $(\Diamond)$ and $(\Box)$ alone.

$(\Diamond)$ Assume that $\{i_1, \ldots, i_k\} : \Diamond\psi, i_1\mathsf{R}j_1, \ldots, i_k\mathsf{R}j_k \in \mathcal{B}_n$. Let $\alpha := \{i_1, \ldots, i_k\}$ and $\beta := \{j_1, \ldots, j_k\}$. We obtain from our assumption that $\mathrm{K}, g[\alpha] \not\models \Diamond\psi$ and $g[\alpha][R]g[\beta]$. From the semantics of $\Diamond$ it follows that $\mathrm{K}, g[\beta] \not\models \psi$. Thus $\mathcal{B}_{n+1} := \mathcal{B}_n \cup \{\beta : \psi\}$ is faithful to K. Clearly $\mathcal{B}_{n+1}$ is an extension of $\mathcal{B}$ by the rule $(\Diamond)$.

$(\Box)$ Assume that $\alpha : \Box\psi \in \mathcal{B}_n$. We obtain from our assumption that $\mathrm{K}, g[\alpha] \not\models \Box\psi$. By the semantics of $\Box$, it follows that $\mathrm{K}, R[g[\alpha]] \not\models \psi$. Now, by Theorem 14, there exists a team $S \subseteq R[g[\alpha]]$ such that $0 < |S| \leq 2^{\mathrm{Rank}_{\mathbb{O}}(\psi)}$ and $\mathrm{K}, S \not\models \psi$. Fix such $S \subseteq R[g[\alpha]]$ and let $u_1, \ldots, u_m$ be the elements of S. Since $S \subseteq R[g[\alpha]]$ there exists a function $h : \{1, \ldots, m\} \to \alpha$ such that $g\big(h(l)\big)Ru_l$, for each $l \leq m$. Let $h' : \{1, \ldots, 2^{\mathrm{Rank}_{\mathbb{O}}(\psi)}\} \to \alpha$ denote the expansion of h defined such that $h'(l) := h(m)$ for $m < l \leq 2^{\mathrm{Rank}_{\mathbb{O}}(\psi)}$. We then extend our function g to a mapping g' to cover new fresh indexes $\beta := \{j_1, \ldots, j_{2^{\mathrm{Rank}_{\mathbb{O}}(\psi)}}\}$. We define that $g'(j_l) := u_l$, for $l \leq m$, and $g'(j_l) := u_m$ for $m < l \leq 2^{\mathrm{Rank}_{\mathbb{O}}(\psi)}$. By construction, we obtain that $\mathrm{K}, g'[\beta] \not\models \psi$ and $g'(h'(l))Rg'(j_l)$ for all $1 \leq l \leq 2^{\mathrm{Rank}_{\mathbb{O}}(\psi)}$. Therefore, together with our assumption, $\mathcal{B}_{n+1} := \mathcal{B}_n \cup \{h'(1)\mathsf{R}j_1, \ldots, h'(2^{\mathrm{Rank}_{\mathbb{O}}(\psi)})\mathsf{R}j_{2^{\mathrm{Rank}_{\mathbb{O}}(\psi)}}, \beta : \psi\}$ is faithful to K by g'. Clearly $\mathcal{B}_{n+1}$ is an extension of $\mathcal{B}$ by the rule $(\Box)$.

Since $\mathcal{T}$ is finite and saturated, $\mathcal{B}_j$ is a saturated branch in $\mathcal{T}$ for some $j \in \mathbb{N}$. Moreover, since $\mathcal{B}_j$ is faithful to K, $\mathcal{B}_j$ is open. ◀

▶ **Theorem 29** (Soundness of $\mathbf{T}_{\mathcal{ML}}$, $\mathbf{T}_{\mathcal{ML}(\mathbb{O})}$, $\mathbf{T}_{\mathcal{MDL}}$, and $\mathbf{T}_{\mathcal{EMDL}}$)**.** *Let $\mathcal{L}$ be a logic in $\{\mathcal{ML}, \mathcal{ML}(\mathbb{O}), \mathcal{MDL}, \mathcal{EMDL}\}$. The calculus $\mathbf{T}_{\mathcal{L}}$ is sound.*

Proof. Fix $\mathcal{L} \in \{\mathcal{ML}, \mathcal{ML}(\mathbb{0}), \mathcal{MDL}, \mathcal{EMDL}\}$. Assume that $\not\models_{\mathcal{L}} \varphi$. By Lemma 28, there is an open saturated branch in every saturated tableau of φ in $\mathbf{T}_{\mathcal{L}}$. Therefore, and since, by Theorem 26, every tableau of φ in $\mathbf{T}_{\mathcal{L}}$ is finite, there does not exists any closed tableau for φ in $\mathbf{T}_{\mathcal{L}}$. Thus $\not\vdash_{\mathbf{T}_{\mathcal{L}}} \varphi$. ◀

▶ **Lemma 30.** *Let $\mathcal{L} \in \{\mathcal{ML}, \mathcal{ML}(\mathbb{0}), \mathcal{MDL}, \mathcal{EMDL}\}$. If there exists an open saturated branch for φ in $\mathbf{T}_{\mathcal{L}}$ then φ is not valid.*

Proof. Let $\mathcal{B}$ be an open saturated branch in a tableau $\mathcal{T}$ of $\mathbf{T}_{\mathcal{L}}$ starting with $\{1, \ldots, 2^{\mathrm{Rank}_{\mathbb{0}}(\varphi)}\} : \varphi$. Define the induced Kripke model $\mathrm{K}_{\mathcal{B}} = (W, R, V)$ from $\mathcal{B}$ as follows: $W := \mathrm{Index}(\mathcal{B})$; iRj iff $i\mathrm{R}j \in \mathcal{B}$; $V(p) := \{i \mid \{i\} : \neg p \in \mathcal{B}\}$ for any p occurring in $\mathcal{B}$, otherwise, $V(p) := \emptyset$. It is straightforward to prove by induction on χ that $\alpha : \chi \in \mathcal{B}$ implies $\mathrm{K}_{\mathcal{B}}, \alpha \not\models \chi$. Since $\{1, \ldots, 2^{\mathrm{Rank}_{\mathbb{0}}(\varphi)}\} : \varphi \in \mathcal{B}$, it follows that $\mathrm{K}_{\mathcal{B}}, \{1, \ldots, 2^{\mathrm{Rank}_{\mathbb{0}}(\varphi)}\} \not\models \varphi$. Thus φ is not valid. ◀

▶ **Theorem 31** (Completeness of $\mathbf{T}_{\mathcal{ML}}$, $\mathbf{T}_{\mathcal{ML}(\mathbb{0})}$, $\mathbf{T}_{\mathcal{MDL}}$, and $\mathbf{T}_{\mathcal{EMDL}}$)**.** *Let $\mathcal{L}$ be a logic in $\{\mathcal{ML}, \mathcal{ML}(\mathbb{0}), \mathcal{MDL}, \mathcal{EMDL}\}$. The calculus $\mathbf{T}_{\mathcal{L}}$ is complete.*

Proof. Fix $\mathcal{L} \in \{\mathcal{ML}, \mathcal{ML}(\mathbb{0}), \mathcal{MDL}, \mathcal{EMDL}\}$. Assume that $\not\vdash_{T_{\mathcal{L}}} \varphi$. Thus every tableau for φ is open. From Theorem 26 it follows that there exists a saturated open tableau for φ. Thus there exists a saturated open branch for φ. Thus, by Lemma 30, $\not\models_{\mathcal{L}} \varphi$. ◀

5 Conclusion

We gave sound and complete Hilbert-style axiomatizations for $\mathcal{PL}$, $\mathcal{PL}(\mathbb{0})$, $\mathcal{PD}$, $\mathcal{ML}(\mathbb{0})$, $\mathcal{MDL}$, and $\mathcal{EMDL}$. In addition, we presented novel labeled tableau calculi for these logics. We proved soundness, completeness and termination for each of the calculi presented.

References

1 Johannes Ebbing, Lauri Hella, Arne Meier, Julian-Steffen Müller, Jonni Virtema, and Heribert Vollmer. Extended modal dependence logic. In *WoLLIC*, pages 126–137, 2013.

2 Johannes Ebbing and Peter Lohmann. Complexity of model checking for modal dependence logic. In Mária Bieliková, Gerhard Friedrich, Georg Gottlob, Stefan Katzenbeisser, and György Turán, editors, *SOFSEM*, volume 7147 of *Lecture Notes in Computer Science*, pages 226–237. Springer, 2012.

3 Johannes Ebbing, Peter Lohmann, and Fan Yang. Model checking for modal intuitionistic dependence logic. In Guram Bezhanishvili, Sebastian Löbner, Vincenzo Marra, and Frank Richter, editors, *Logic, Language, and Computation*, volume 7758 of *Lecture Notes in Computer Science*, pages 231–256. Springer, 2013.

4 Lauri Hella, Kerkko Luosto, Katsuhiko Sano, and Jonni Virtema. The expressive power of modal dependence logic. In *AiML 2014*, 2014.

5 Jarmo Kontinen. *Coherence and Complexity in Fragments of Dependence Logic*. PhD thesis, University of Amsterdam, 2010.

6 Juha Kontinen, Julian-Steffen Müller, Henning Schnoor, and Heribert Vollmer. A van benthem theorem for modal team semantics. *CSL*, 2015.

7 Katsuhiko Sano and Jonni Virtema. Characterizing frame definability in team semantics via the universal modality. In Valeria de Paiva, Ruy de Queiroz, Lawrence S. Moss, Daniel Leivant, and Anjolina G. de Oliveira, editors, *Logic, Language, Information, and Computation*, volume 9160 of *Lecture Notes in Computer Science*, pages 140–155. Springer Berlin Heidelberg, 2015.

8 Merlijn Sevenster. Model-theoretic and computational properties of modal dependence logic. *J. Log. Comput.*, 19(6):1157–1173, 2009.

9 Jouko Väänänen. *Dependence Logic – A New Approach to Independence Friendly Logic*, volume 70 of *London Mathematical Society student texts*. Cambridge University Press, 2007.

10 Jouko Väänänen. Modal dependence logic. In Krzysztof R. Apt and Robert van Rooij, editors, *New Perspectives on Games and Interaction*, volume 4 of *Texts in Logic and Games*, pages 237–254. Amsterdam University Press, 2008.

11 Jonni Virtema. Complexity of validity for propositional dependence logics. In *GandALF 2014*, 2014.

12 Fan Yang. *On Extensions and Variants of Dependence Logic*. PhD thesis, University of Helsinki, 2014.

13 Fan Yang and Jouko A. Väänänen. Propositional logics of dependence and independence, part i. *arXiv:1412.7998*, 2014.

Static Analysis for Logic-based Dynamic Programs*

Thomas Schwentick, Nils Vortmeier, and Thomas Zeume

TU Dortmund University
Germany
{thomas.schwentick,nils.vortmeier,thomas.zeume}@tu-dortmund.de

Abstract

A dynamic program, as introduced by Patnaik and Immerman (1994), maintains the result of a fixed query for an input database which is subject to tuple insertions and deletions. It can use an auxiliary database whose relations are updated via first-order formulas upon modifications of the input database. This paper studies static analysis problems for dynamic programs and investigates, more specifically, the decidability of the following three questions. Is the answer relation of a given dynamic program always empty? Does a program actually maintain a query? Is the content of auxiliary relations independent of the modification sequence that lead to an input database? In general, all these problems can easily be seen to be undecidable for full first-order programs. Therefore the paper aims at pinpointing the exact decidability borderline for programs with restricted arity (of the input and/or auxiliary database) and restricted quantification.

1998 ACM Subject Classification F.4.1. Mathematical Logic

Keywords and phrases Dynamic descriptive complexity, algorithmic problems, emptiness, history independence, consistency

Digital Object Identifier 10.4230/LIPIcs.CSL.2015.308

1 Introduction

In modern database scenarios data is subject to frequent changes. In order to avoid costly re-computation of queries from scratch after each small modification of the data, one can try to use previously computed auxiliary data. This auxiliary data then needs to be updated dynamically whenever the database changes.

The descriptive dynamic complexity framework (short: dynamic complexity) by Patnaik and Immerman [19] models this setting from a declarative perspective. It was mainly inspired by updates in relational databases. Within this framework, for a relational database subject to change, a *dynamic program* maintains auxiliary relations with the intention to help answering a query $\mathcal{Q}$. When a modification to the database, that is an insertion or deletion of a tuple, occurs, every auxiliary relation is updated through a first-order update formula (or, equivalently, through a core SQL query) that can refer to the database as well as to the auxiliary relations. The result of $\mathcal{Q}$ is, at every time, represented by some distinguished auxiliary relation. The class of all queries maintainable by dynamic programs with first-order update formulas is called DynFO and we refer to such programs as DynFO-programs. We note that shortly before the work of Patnaik and Immerman, the declarative approach was independently formalized in a similar way by Dong, Su and Topor [6].

* The first and third author acknowledge the financial support by DFG grant SCHW 678/6-1.

The main question studied in Dynamic Complexity has been which queries that are not statically expressible in first-order logic (and therefore not in Core SQL), can be maintained by DynFO-programs. Recently, it has been shown that the Reachability query, a very natural such query, can be maintained by DynFO programs [1]. Altogether, research in Dynamic Complexity succeeded in proving that many non-FO queries are maintainable in DynFO. These results and their underlying techniques yield many interesting insights into the the nature of Dynamic Complexity.

However, to complete the understanding of Dynamic Complexity, it would be desirable to complement these techniques by methods for proving that certain queries are *not* maintainable by DynFO programs. But the state of the art with respect to inexpressibility results is much less favorable: at this point, no general techniques for showing that a query is not expressible in DynFO are available. In order to get a better overall picture of Dynamic Complexity in general and to develop methods for inexpressibility proofs in particular, various restrictions of DynFO have been studied, based on, e.g., arity restrictions for the auxiliary relations [2, 5, 3], fragments of first-order logic [12, 10, 25, 23], or by other means [4, 11].

At the heart of our difficulties to prove inexpressibility results in Dynamic Complexity is our limited understanding of what dynamic programs with or without restrictions "can do" in general, and our limited ability to analyze what a particular dynamic program at hand "does". In this paper, we initiate a systematic study of the "analyzability" of dynamic programs. Static analysis of queries has a long tradition in Database Theory and we follow this tradition by first studying the emptiness problem for dynamic programs, that is the question, whether there exists an initial database and a modification sequence that is accepted by a given dynamic program.[1] Given the well-known undecidability of the finite satisfiability problem for first-order logic [21], it is not surprising that emptiness of DynFO programs is undecidable in general. However, we try to pinpoint the borderline of undecidability for fragments of DynFO based on restrictions of the arity of input relations, the arity of auxiliary relations and for the class DynProp of programs with quantifier-free update formulas.

In the fragments where undecidability of emptiness does not directly follow from undecidability of satisfiability in the corresponding fragment of first-order logic, our undecidability proofs make use of dynamic programs whose query answer might not only depend on the database yielded by a certain modification sequence, but also on the sequence itself, that is, on the order in which tuples are inserted or (even) deleted. From a useful dynamic program one would, of course, expect that it is *consistent* in the sense that its query answer always only depends on the current database, but not on the specific modification sequence by which it has been obtained. It turns out that the emptiness problem for consistent programs is easier than the general emptiness problem for dynamic programs. More precisely, there are fragments of DynFO, for which an algorithm can decide emptiness for dynamic programs that come with a "consistency guarantee", but for which the emptiness problem is undecidable, in general. However, it turns out that the combination of a consistency test with an emptiness test for consistent programs does not gain any advantage over "direct" emptiness tests, since the consistency problem turns out to be as difficult as the general emptiness problem.

Finally, we study a property that many dynamic programs in the literature share: they are *history independent* in the sense that all auxiliary relations always only depend on the

[1] The exact framework will be defined in Section 3, but we already mention that we will consider the setting in which databases are initially empty and the auxiliary relations are defined by first-order formulas.

Table 1 Summary of the results of this paper. DynFO(ℓ-in, m-aux) stands for DynFO-programs with (at most) ℓ-ary input relations and m-ary auxiliary relations. DynFO(m-aux) and DynFO(ℓ-in) represent programs with m-ary auxiliary relations (and arbitrary input relations) and programs with ℓ-ary input relations, respectively. Likewise for DynProp.

	Emptiness Consistency	Emptiness for consistent programs	History Independence
Undecidable	DynFO(1-in, 0-aux) DynProp(2-in, 0-aux) DynProp(1-in, 2-aux)	DynFO(1-in, 2-aux) DynFO(2-in, 0-aux)	DynFO(2-in, 0-aux)
Decidable	DynProp(1-in, 1-aux)	DynFO(1-in, 1-aux) DynProp(1-in) DynProp(1-aux)	DynFO(1-in) DynProp(1-aux)
Open		DynProp(2-in, 2-aux) and beyond	DynProp(2-in, 2-aux) and beyond

current (input) database. History independence can be seen as a strong form of consistency in that it not only requires the query relation, but *all* auxiliary relations to be determined by the input database. History independent dynamic programs (also called *memoryless* [19] or *deterministic* [4]) are still expressive enough to maintain interesting queries like undirected reachability [11]. But also some inexpressibility proofs have been found for such programs [4, 11, 25]. We study the *history independence problem*, that is, whether a given dynamic program is history independent. In a nutshell, the history independence problem is the "easiest" of the static analysis problems considered in this paper.

Our results, summarized in Table 1, shed light on the borderline between decidable and undecidable fragments of DynFO with respect to emptiness (and consistency), emptiness for consistent programs and history independence. While the picture is quite complete for the emptiness problem for general dynamic programs, for some fragments of DynProp there remain open questions regarding the emptiness problem for consistent dynamic programs and the history-independence problem. Some of the results shown in this paper have been already presented in the master's thesis of Nils Vortmeier [22].

Outline. We recall some basic definitions in Section 2 and introduce the formal setting in Section 3. The emptiness problem is defined and studied in Section 4, where we first consider general dynamic programs (Subsection 4.1) and then consistent dynamic programs (Subsection 4.2). In Subsection 4.3 we briefly discuss the impact of built-in orders to the results. The Consistency and History Independence problems are studied in Sections 5 and 6, respectively. We conclude in Section 7. Due to the space limit we only give proof sketches or even proof ideas in the body of this paper. Complete proofs can be found in the full version [20].

2 Preliminaries

We presume that the reader is familiar with basic notions from Finite Model Theory and refer to [8, 16] for a detailed introduction into this field. We review some basic definitions in order to fix notations.

In this paper, a *domain* is a non-empty finite set. For tuples $\vec{a} = (a_1, \ldots, a_k)$ and $\vec{b} = (b_1, \ldots, b_\ell)$ over some domain D, the $(k + \ell)$-tuple obtained by concatenating $\vec{a}$ and $\vec{b}$ is denoted by $(\vec{a}, \vec{b})$.

A (relational) *schema* is a collection τ of relation symbols[2] together with an arity function $\mathrm{Ar} : \tau \to \mathbb{N}$. A *database* $\mathcal{D}$ with schema τ and domain D is a mapping that assigns to every relation symbol $R \in \tau$ a relation of arity $\mathrm{Ar}(R)$ over D. The *size of a database*, usually denoted by n, is the size of its domain. We call a database *empty*, if all its relations are empty. We emphasize that empty databases have non-empty domains. A *τ-structure* $\mathcal{S}$ is a pair $(D, \mathcal{D})$ where $\mathcal{D}$ is a database with schema τ and domain D. Often we omit the schema when it is clear from the context.

We write $\mathcal{S} \models \varphi(\vec{a})$ if the first-order formula $\varphi(\vec{x})$ holds in $\mathcal{S}$ under the variable assignment that maps $\vec{x}$ to $\vec{a}$. The *quantifier depth* of a first-order formula is the maximal nesting depth of quantifiers. The *rank-q type* of a tuple $(a_1, \ldots, a_m)$ with respect to a τ-structure $\mathcal{S}$ is the set of all first-order formulas $\varphi(x_1, \ldots, x_m)$ (with equality) of quantifier depth at most q, for which $\mathcal{S} \models \varphi(\vec{a})$ holds. By $\mathcal{S} \equiv_q \mathcal{S}'$ we denote that two structures $\mathcal{S}$ and $\mathcal{S}'$ have the same rank-q type (of length 0 tuples).

For a subschema $\tau' \subseteq \tau$, the rank-q τ'-type of a tuple $\vec{a}$ in a τ-structure $\mathcal{S}$ is its rank-q type in the τ'-reduct of $\mathcal{S}$.

We refer to the rank-0 type of a tuple also as its *atomic type* and, since we mostly deal with rank-0 types, simply as its *type*. The *equality type* of a tuple is the atomic type with respect to the empty schema.

The *k-ary type* of a tuple $\vec{a}$ in a structure $\mathcal{S}$ is its $\tau_{\leq k}$-type, where $\tau_{\leq k}$ consists of all relation symbols of τ with arity at most k. The *τ'-color* of an element a in $\mathcal{S}$, for a subschema τ' of the schema of $\mathcal{S}$, is its τ'_1-type, where τ'_1 consists of all unary relation symbols of τ'. We often enumerate the possible τ'-colors as $c_0, \ldots, c_L$, for some L with c_0 being the color of elements that are in neither of the unary relations. We call these elements *τ'-uncolored*. If τ' is clear from the context we simply speak of colors and uncolored elements.

3 The dynamic complexity setting

For a database $\mathcal{D}$ over schema τ, a *modification* $\delta = (o, \vec{a})$ consists of an operation $o \in \{\mathrm{INS}_S, \mathrm{DEL}_S \mid S \in \tau\}$ and a tuple $\vec{a}$ of elements from the domain of $\mathcal{D}$. By $\delta(\mathcal{D})$ we denote the result of applying δ to $\mathcal{D}$ with the obvious semantics of inserting or deleting the tuple $\vec{a}$ to or from relation $S^{\mathcal{D}}$. For a sequence $\alpha = \delta_1 \cdots \delta_N$ of modifications to a database $\mathcal{D}$ we let $\alpha(\mathcal{D}) \stackrel{\text{def}}{=} \delta_N(\cdots(\delta_1(\mathcal{D}))\cdots)$.

A *dynamic instance*[3] of a query $\mathcal{Q}$ is a pair $(\mathcal{D}, \alpha)$, where $\mathcal{D}$ is a database over a domain D and α is a sequence of modifications to $\mathcal{D}$. The dynamic query $\mathrm{DYN}(\mathcal{Q})$ yields the result of evaluating the query $\mathcal{Q}$ on $\alpha(\mathcal{D})$.

Dynamic programs, to be defined next, consist of an initialization mechanism and an update program. The former yields, for every (initial) database $\mathcal{D}$, an initial state with initial auxiliary data. The latter defines the new state of the dynamic program for each possible modification δ.

A *dynamic schema* is a pair $(\tau_{\text{in}}, \tau_{\text{aux}})$, where τ_{in} and τ_{aux} are the schemas of the input database and the auxiliary database, respectively. We call relations over τ_{in} *input relations* and relations over τ_{aux} *auxiliary relations*. If the relations are 0-ary, we also speak of input or auxiliary *bits*. We always let $\tau \stackrel{\text{def}}{=} \tau_{\text{in}} \cup \tau_{\text{aux}}$.

[2] For simplicity we do not allow constants in this work but note that our results hold for relational schemas with constants as well.

[3] The following introduction to dynamic descriptive complexity is similar to previous work [25, 24].

▶ **Definition 1** (Update program). An *update program* P over a dynamic schema $(\tau_{\text{in}}, \tau_{\text{aux}})$ is a set of first-order formulas (called *update formulas* in the following) that contains, for every $R \in \tau_{\text{aux}}$ and every $o \in \{\text{INS}_S, \text{DEL}_S \mid S \in \tau_{\text{in}}\}$, an update formula $\phi_o^R(\vec{x}; \vec{y})$ over the schema τ where $\vec{x}$ and $\vec{y}$ have the same arity as S and R, respectively.

A *program state* $\mathcal{S}$ over dynamic schema $(\tau_{\text{in}}, \tau_{\text{aux}})$ is a structure $(D, \mathcal{I}, \mathcal{A})$ where[4] D is a finite domain, $\mathcal{I}$ is a database over the input schema (the *input database*) and $\mathcal{A}$ is a database over the auxiliary schema (the *auxiliary database*). The *semantics of update programs* is as follows. For a modification $\delta = (o, \vec{a})$, where $\vec{a}$ is a tuple over D, and program state $\mathcal{S} = (D, \mathcal{I}, \mathcal{A})$ we denote by $P_\delta(\mathcal{S})$ the state $(D, \delta(\mathcal{I}), \mathcal{A}')$, where $\mathcal{A}'$ consists of relations $R^{\mathcal{A}'} \stackrel{\text{def}}{=} \{\vec{b} \mid \mathcal{S} \models \phi_o^R(\vec{a}; \vec{b})\}$. The effect $P_\alpha(\mathcal{S})$ of a modification sequence $\alpha = \delta_1 \dots \delta_N$ to a state $\mathcal{S}$ is the state $P_{\delta_N}(\dots(P_{\delta_1}(\mathcal{S}))\dots)$.

▶ **Definition 2** (Dynamic program). A *dynamic program* is a triple $(P, \text{Init}, R_{\mathcal{Q}})$, where
- P is an update program over some dynamic schema $(\tau_{\text{in}}, \tau_{\text{aux}})$,
- Init is a mapping that maps τ_{in}-databases to τ_{aux}-databases, and
- $R_{\mathcal{Q}} \in \tau_{\text{aux}}$ is a designated *query symbol*.

A dynamic program $\mathcal{P} = (P, \text{Init}, R_{\mathcal{Q}})$ *maintains* a dynamic query $\text{Dyn}(\mathcal{Q})$ if, for every dynamic instance $(\mathcal{D}, \alpha)$, the query result $\mathcal{Q}(\alpha(\mathcal{D}))$ coincides with the query relation $R_{\mathcal{Q}}^{\mathcal{S}}$ in the state $\mathcal{S} = P_\alpha(\mathcal{S}_{\text{Init}}(\mathcal{D}))$, where $\mathcal{S}_{\text{Init}}(\mathcal{D}) \stackrel{\text{def}}{=} (D, \mathcal{D}, \text{Init}(\mathcal{D}))$ is the initial state for $\mathcal{D}$. If the query relation $R_{\mathcal{Q}}$ is 0-ary, we often denote this relation as *query bit* Acc and say that $\mathcal{P}$ *accepts* α over D if Acc is true in $P_\alpha(\mathcal{S}_{\text{Init}}(\mathcal{D}))$.

In the following, we write $\mathcal{P}_\alpha(\mathcal{D})$ instead of $P_\alpha(\mathcal{S}_{\text{Init}}(\mathcal{D}))$ and $\mathcal{P}_\alpha(\mathcal{S})$ instead[5] of $P_\alpha(\mathcal{S})$ for a given dynamic program $\mathcal{P} = (P, \text{Init}, R_{\mathcal{Q}})$, a modification sequence α, an initial database $\mathcal{D}$ and a state $\mathcal{S}$.

▶ **Definition 3** (DynFO and DynProp). DynFO is the class of all dynamic queries that can be maintained by dynamic programs with first-order update formulas and first-order definable initialization mapping when starting from an initially empty input database. DynProp is the subclass of DynFO, where update formulas are quantifier-free[6].

A DynFO-program is a dynamic program with first-order update formulas, likewise a DynProp-program is a dynamic program with quantifier-free update formulas. A DynFO(ℓ-in, m-aux)-program is a DynFO-program over (at most) ℓ-ary input databases that uses auxiliary relations of arity at most m; likewise for DynProp(ℓ-in, m-aux)-programs.[7]

Due to the undecidability of finite satisfiability of first-order logic, the emptiness problem – the problem we study first – is undecidable even for DynFO-programs with only a single auxiliary relation (more precisely, with query bit only). Therefore, we restrict our investigations to fragments of DynFO. Also allowing arbitrary initialization mappings immediately yields an undecidable emptiness problem. This is already the case for first-order definable initialization mappings for arbitrary initial databases. In the literature classes with various restricted and unrestricted initialization mappings have been studied, see [24] for a discussion. In this work, in line with [19], we allow initialization mappings defined by arbitrary first-order formulas, but require that the initial database is empty. Of course, we could have studied

[4] We prefer the notation $(D, \mathcal{I}, \mathcal{A})$ over $(D, \mathcal{I} \cup \mathcal{A})$ to emphasize the two components of the overall database.
[5] The notational difference is tiny here: we refer to the dynamic program instead of the update program.
[6] We still allow the use of quantifiers for the initialization.
[7] We do not consider the case $\ell = 0$ where databases are pure sets with a fixed number of bits.

further restrictions on the power of the initialization formulas, but this would have yielded a setting with an additional parameter.

The following example illustrates a technique to maintain lists with quantifier-free dynamic programs, introduced in [10, Proposition 4.5], which is used in some of our proofs. The example itself is from [25].

▶ **Example 4.** We provide a DynProp-program $\mathcal{P}$ for the dynamic variant of the Boolean query NonEmptySet, where, for a unary relation U subject to insertions and deletions of elements, one asks whether U is empty. Of course, this query is trivially expressible in first-order logic, but not without quantifiers.

The program $\mathcal{P}$ is over auxiliary schema $\tau_{\text{aux}} = \{R_Q, \text{First}, \text{Last}, \text{List}\}$, where R_Q is the query bit (i.e. a 0-ary relation symbol), First and Last are unary relation symbols, and List is a binary relation symbol. The idea of $\mathcal{P}$ is to maintain a list of all elements currently in U. The list structure is stored in the binary relation $\text{List}^{\mathcal{S}}$. The first and last element of the list are stored in $\text{First}^{\mathcal{S}}$ and $\text{Last}^{\mathcal{S}}$, respectively. We note that the order in which the elements of U are stored in the list depends on the order in which they are inserted into U.

For a given instance of NonEmptySet the initialization mapping initializes the auxiliary relations accordingly.We only describe the (more complicated) case of deletions from U.

Deletion of a from U. How a deleted element a is removed from the list, depends on whether a is the first element of the list, the last element of the list or some other element of the list. The query bit remains 'true', if a was not the first *and* last element of the list.[8]

$$\begin{aligned}
\phi^{\text{First}}_{\text{DEL}_U}(a;x) &\stackrel{\text{def}}{=} (\text{First}(x) \wedge x \neq a) \vee (\text{First}(a) \wedge \text{List}(a,x)) \\
\phi^{\text{Last}}_{\text{DEL}_U}(a;x) &\stackrel{\text{def}}{=} (\text{Last}(x) \wedge x \neq a) \vee (\text{Last}(a) \wedge \text{List}(x,a)) \\
\phi^{\text{List}}_{\text{DEL}_U}(a;x,y) &\stackrel{\text{def}}{=} x \neq a \wedge y \neq a \wedge \big(\text{List}(x,y) \vee (\text{List}(x,a) \wedge \text{List}(a,y))\big) \\
\phi^{R_Q}_{\text{DEL}_U}(a) &\stackrel{\text{def}}{=} \neg(\text{First}(a) \wedge \text{Last}(a))
\end{aligned}$$

◀

In some parts of the paper we will use specific forms of modification sequences. An *insertion sequence* is a modification sequence $\alpha = \delta_1 \cdots \delta_m$ whose modifications are pairwise distinct insertions. An insertion sequence α over a unary input schema τ_{in} is in *normal form* if it fulfills the following two conditions.

(N1) For each element a, the insertions affecting a form a contiguous subsequence α_a of α. We say that α_a *colors* a.

(N2) For all elements a, b that get assigned the same τ_{in}-color by α, the projections of the subsequences α_a and α_b to their operations (i.e., their first parameters) are identical.

4 The Emptiness Problem

In this section we define and study the decidability of the emptiness problem for dynamic programs in general and for restricted classes of dynamic programs. The emptiness problem asks, whether the query relation R_Q of a given dynamic program $\mathcal{P}$ is always empty, more precisely, whether $R^{\mathcal{S}}_Q = \emptyset$ for every (empty) initial database $\mathcal{D}$ and every modification sequence α with $\mathcal{S} = \mathcal{P}_\alpha(\mathcal{D})$.

[8] We omit the (obvious) parts of formulas that deal with spurious deletions.

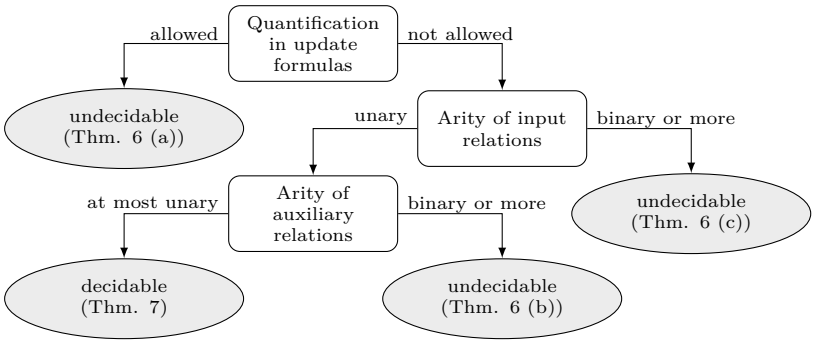


Figure 1 Decidability of EMPTINESS for various classes of dynamic programs.

To enable a fine-grained analysis, we parameterize the emptiness problem by a class $\mathcal{C}$ of dynamic programs.

Problem:	EMPTINESS($\mathcal{C}$)
Input:	A dynamic program $\mathcal{P} \in \mathcal{C}$ with FO initialization
Question:	Is $R_Q^{\mathcal{S}} = \emptyset$, for every initially empty database $\mathcal{D}$ and every modification sequence α, where $\mathcal{S} \stackrel{\text{def}}{=} \mathcal{P}_\alpha(\mathcal{D})$?

As mentioned before, undecidability of the emptiness problem for unrestricted dynamic programs follows immediately from the undecidability of finite satisfiability of first-order logic.

▶ **Theorem 5.** EMPTINESS *is undecidable for* DYNFO(2-*in*, 0-*aux*)*-programs.*

In the remainder of this section, we will shed some light on the border line between decidable and undecidable fragments of DYNFO. In Subsection 4.1 we study fragments of DYNFO obtained by disallowing quantification and/or restricting the arity of input and auxiliary relations. In Subsection 4.2, we consider dynamic programs that come with a certain consistency guarantee.

4.1 Emptiness of general dynamic programs

In this subsection we study the emptiness problem for various restricted classes of dynamic programs. We will see that the problem is basically only decidable if all relations are at most unary and no quantification in update formulas is allowed. Figure 1 summarizes the results.

At first we strengthen the general result from Theorem 5. We show that undecidability of the emptiness problem for DYNFO-programs holds even for unary input relations and auxiliary bits. Furthermore, quantification is not needed to yield undecidability: for DYNPROP-programs, emptiness is undecidable for binary input or auxiliary relations.

▶ **Theorem 6.** *The emptiness problem is undecidable for*
(a) DYNFO(1-*in*, 0-*aux*)*-programs,*
(b) DYNPROP(1-*in*, 2-*aux*)*-programs,*
(c) DYNPROP(2-*in*, 0-*aux*)*-programs,*

Proof sketch. In all three cases, the proof is by a reduction from the emptiness problem for semi-deterministic 2-counter automata.

In a nutshell, a counter automaton (short: CA) is a finite automaton that is equipped with counters that range over the non-negative integer numbers. A counter c can be incremented

(inc(c)), decremented (dec(c)) and tested for zero (ifzero(c)). A CA does not read any input (i.e., its transitions can be considered to be ϵ-transitions) and in each step it can manipulate or test one counter and transit from one state to another state. More formally, a CA is tuple (Q, C, Δ, q_i, F), where Q is a set of states, $q_i \in Q$ is the initial state, $F \subseteq Q$ is the set of accepting states, and C is a finite set (the *counters*). The transition relation Δ is a subset of $Q \times \{\text{inc}(c), \text{dec}(c), \text{ifzero}(c) \mid c \in C\} \times Q$. A *configuration* of a CA is a pair $(p, \vec{n})$ where p is a state and $\vec{n} \in \mathbb{N}^C$ gives a value n_c for each counter c in C. A transition $(p, \text{inc}(c), q)$ can be applied in state p, transits to state q and increments n_c by one. A transition $(p, \text{dec}(c), q)$ can be applied in state p if $n_c > 0$, transits to state q and decrements n_c by one. A transition $(p, \text{ifzero}(c), q)$ can be applied in state p, if $n_c = 0$ and transits to state q.

A CA is *semi-deterministic* if from every state there is either at most one transition or there are two transitions, one decrementing and one testing the same counter for zero. The emptiness problem for (semi-deterministic) 2-counter automata (2CA) asks whether a given counter automaton with two counters has an accepting run and is undecidable [18, Theorem 14.1-1].

In all three reductions, the dynamic program $\mathcal{P}$ is constructed such that for every run ρ of a semi-deterministic 2CA $\mathcal{M}$ there is a modification sequence $\alpha = \alpha(\rho)$ that lets $\mathcal{P}$ simulate ρ, and an empty database $\mathcal{D}$, such that $\mathcal{P}$ accepts α over $\mathcal{D}$ if and only if ρ is accepting. More precisely, the states of $\mathcal{P}$ encode the states of $\mathcal{M}$ by auxiliary bits and the counters of $\mathcal{M}$ in some way that differs in the three cases. However, in all cases it holds that not every modification sequence for $\mathcal{P}$ corresponds to a run of $\mathcal{M}$. However, $\mathcal{P}$ can detect if α does *not* correspond to a run and assume a rejecting sink state as soon as this happens.

For (a), the two counters are simply represented by two unary relations, such that the number of elements in a relation is the current value of the counter. The test whether a counter has value zero thus boils down to testing emptiness of a set and can easily be expressed by a formula with quantifiers.

The lack of quantifiers makes the reductions for (b) and (c) a bit more complicated. In both cases, the counters are represented by linked lists, similar to Example 4, and the number of elements in the list corresponds to the counter value (in (c): plus 1). With such a list a counter value zero can be detected without quantification. Due to the allowed relation types, the lists are built with auxiliary relations in (b) and with input relations in (c). ◀

The next result shows that emptiness of DynProp(1-in, 1-aux)-programs is decidable, yielding a clean boundary between decidable and undecidable fragments.

▶ **Theorem 7.** Emptiness *is decidable for* DynProp(1-*in*, 1-*aux*)-*programs.*

Proof. The proof uses the following two simple observations about DynProp(1-in, 1-aux)-programs $\mathcal{P}$.

- The initialization formulas of $\mathcal{P}$ assign the same τ_{aux}-color to all elements. This color and the initial auxiliary bits only depend on the size of the domain. Furthermore there is a number $n(\mathcal{P})$, depending solely on the initialization formulas, such that the initial auxiliary bits and τ_{aux}-colors are the same for all empty databases with at least $n(\mathcal{P})$ elements. This observation actually also holds for DynFO(1-in, 1-aux)-programs.
- When $\mathcal{P}$ reacts to a modification $\delta = (o, a)$, the new (τ-)color of an element $b \neq a$ only depends on o, the old color of b, the old color of a, and the 0-ary relations. In particular, if two elements b_1, b_2 (different from a) have the same color before the update, they both have the same new color after the update. Thus, the overall update basically consists of assigning new colors to each color (for all elements except a), and the appropriate handling of the element a and the 0-ary relations.

We will show below that the behavior of DynProp(1-in, 1-aux)-programs can be simulated by an automaton model with a decidable emptiness problem, which we introduce next.

A *multicounter automaton* (short: MCA) is a counter automaton which is not allowed to test whether a counter is zero, i.e. the transition relation Δ is a subset of $Q \times \{\text{inc}(c), \text{dec}(c) \mid c \in C\} \times Q$. A *transfer multicounter automaton* (short: TMCA) is a multicounter counter automaton which has, in addition to the increment and the decrement operation, an operation that simultaneously transfers the content of each counter to another counter. More precisely the transition relation Δ is a subset of $Q \times (\{\text{inc}(c), \text{dec}(c) \mid c \in C\} \cup \{t \mid t : C \to C\}) \times Q$. Applying a transition (p, t, q) to a configuration $(p, \vec{n})$ yields a configuration $(q, \vec{n}')$ with $n'_c \stackrel{\text{def}}{=} \sum_{t(d)=c} n_d$ for every $c \in C$. A configuration $(q, \vec{n})$ of a TCMA is *accepting*, if $q \in F$. The emptiness problem for TCMAs[9] is decidable by reduction to the coverability problem for transfer petri nets[10] which is known to be decidable [7].

Let $\mathcal{P}$ be a DynProp-program over unary schema $\tau = \tau_{\text{in}} \cup \tau_{\text{aux}}$ with query symbol $R_{\mathcal{Q}}$ which may be 0-ary or unary. Let Γ_0 be the set of all 0-ary (atomic) types over τ and let Γ_1 be the set of τ-colors. We construct a transfer multicounter automaton $\mathcal{M}$ with counter set $Z_1 = \{z_\gamma \mid \gamma \in \Gamma_1\}$. The state set Q of $\mathcal{M}$ contains Γ_0, the only accepting state f and some further "intermediate" states to be specified below.

The intuition is that whenever $\mathcal{P}$ can reach a state $\mathcal{S}$ then $\mathcal{M}$ can reach a configuration $c = (p, \vec{n})$ such that p reflects the 0-ary relations in $\mathcal{S}$ and, for every $\gamma \in \Gamma_1$, n_γ is the number of elements of color γ in $\mathcal{S}$.

The automaton $\mathcal{M}$ works in two phases. First, $\mathcal{M}$ guesses the size n of the domain of the initial database. To this end, it increments the counter z_γ to n, where γ is the color assigned to all elements by the initialization formula for domains of size n, and it assumes the state corresponding to the initial 0-ary relations for a database of size n. Here the first of the above observations is used. Then $\mathcal{M}$ simulates an actual computation of $\mathcal{P}$ from the initial database of size n as follows. Every modification $\text{INS}_S(a)$ (or $\text{DEL}_S(a)$, respectively) in $\mathcal{P}$ is simulated by a sequence of three transitions in $\mathcal{M}$:

- First, the counter z_γ, where γ is the color of a before the modification, is decremented.
- Second, the counters for all colors are adapted according to the update formulas of $\mathcal{P}$.
- Third, the counter $z_{\gamma'}$, where γ' is the color of a after the modification, is incremented.

If a modification changes an input bit, the first and third step are omitted. The state of $\mathcal{M}$ is changed to reflect the changes of the 0-ary relations of $\mathcal{P}$. For this second phase the second of the above observations is used.

To detect when the simulation of $\mathcal{P}$ reaches a state with non-empty query relation $R_{\mathcal{Q}}$, states $p \in \Gamma_0$ may have a transition to the accepting state f. ◀

4.2 Emptiness of consistent dynamic programs

Some readers of the proof of Theorem 6 might have got the impression that we were cheating a bit, since the dynamic programs it constructs do not behave as one would expect: in all three cases each modification sequence α that yields a non-empty query relation $R_{\mathcal{Q}}$ can be changed, e.g., by switching two operations, into a sequence that does not correspond to a run of the CA and therefore does *not* yield a non-empty query relation. That is, the program $\mathcal{P}$ is *inconsistent* because it might yield different results when the same database is reached through two different modification sequences.

---

[9] We note that (the complement of) this emptiness problem is often called *control-state reachability* problem.

[10] The simulation of states by counters can be done as in [13, Lemma 2.1]

It seems, that this inconsistency made the proof of Theorem 6 much easier. Therefore, the question arises, whether the emptiness problem becomes easier if it can be taken for granted that the given dynamic program is actually consistent. We study this question in this subsection and will investigate the related decision problem whether a given dynamic program is consistent in the next section.

As Table 1 shows, the emptiness problem for consistent dynamic programs is indeed easier in the sense that it is decidable for a considerably larger class of dynamic programs. While emptiness for general DynFO programs is already undecidable for the tiny fragment with unary input relations and 0-ary auxiliary relations, it is decidable for consistent DynFO programs with unary input and unary auxiliary relations. Likewise, for DynProp there is a significant gap: for consistent programs it is decidable for arbitrary input arities (with unary auxiliary relations) or arbitrary auxiliary arities (with unary input relations), but for general programs emptiness becomes undecidable as soon as binary relations are available (in the input *or* in the auxiliary database).

We call a dynamic program $\mathcal{P}$ *consistent*, if it maintains a query with respect to an empty initial database, that is, if, for all modification sequences α to an empty initial database $\mathcal{D}_\emptyset$, the query relation in $\mathcal{P}_\alpha(\mathcal{D}_\emptyset)$ depends only on the database $\alpha(\mathcal{D}_\emptyset)$. In the remainder of this subsection we show the undecidability and decidability results stated in Table 1.

▶ **Theorem 8.** *The emptiness problem is undecidable for*
(a) *consistent* DynFO(2-*in*, 0-*aux*)*-programs, and*
(b) *consistent* DynFO(1-*in*, 2-*aux*)*-programs.*

Proof idea. Statement (a) is a corollary of the proof of Theorem 5, as the reduction in that proof always yields a consistent program.

For (b), we present another reduction from the emptiness problem for semi-deterministic 2CAs (see also the proof of Theorem 6). From a semi-deterministic 2CA $\mathcal{M}$ we will construct a consistent Boolean dynamic program $\mathcal{P}$ with a single unary input relation U. The query maintained by $\mathcal{P}$ is "$\mathcal{M}$ halts after at most $|U|$ steps". Clearly, such a program has a non-empty query result for some database and some modification sequence if and only if $\mathcal{M}$ has an accepting run.

The general idea is that $\mathcal{P}$ simulates one step of the run of $\mathcal{M}$ whenever a new element is inserted to U. A slight complication arises from deletions from U, since it is not clear how one could simulate $\mathcal{M}$ one step "backwards". Therefore, when an element is deleted from U, $\mathcal{P}$ freezes the simulation and stores the size m of $|U|$ before the deletion. It continues the simulation as soon as the current size ℓ of U grows larger than m, for the first time. ◀

Contrary to the case of not necessarily consistent programs, the emptiness problem is decidable for consistent DynFO(1-in, 1-aux)-programs.

▶ **Theorem 9.** Emptiness *is decidable for consistent* DynFO(1-*in*, 1-*aux*)*-programs.*

Proof idea. The proof uses the fact that the truth of first-order formulas with quantifier depth k in a state of a DynFO(1-in, 1-aux)-program only depends on the number of elements of every color up to k. The states of a consistent DynFO(1-in, 1-aux)-program can therefore be abstracted by a bounded amount of information, namely the number of elements of every color up to $k+1$. This can be used to construct, from a consistent DynFO(1-in, 1-aux)-program $\mathcal{P}$, a nondeterministic finite automaton $\mathcal{A}$ that reads encoded modification sequences for $\mathcal{P}$ in normal form and represents the abstracted state of $\mathcal{P}$ in its own state. In this way the emptiness problem for consistent DynFO(1-in, 1-aux)-programs reduces to the emptiness problem for nondeterministic finite automata. ◀

The picture of decidability of emptiness for consistent programs for all classes of the form DynFO(ℓ-in, m-aux) is pretty clear and simple: it is decidable if and only if $\ell = 1$ *and* $m \leq 1$. Now we turn our focus to the corresponding classes of consistent DynProp-programs. Here we do not have a full picture. We show in the following that it is decidable if $\ell = 1$ *or* $m \leq 1$.

▶ **Theorem 10.** *The emptiness problem is decidable for*
(a) *consistent* DynProp(1-*in*)-*programs.*
(b) *consistent* DynProp(1-*aux*)-*programs.*

Proof idea (of Theorem 10 (a)). The statement follows almost immediately from the fact that every consistent DynProp(1-in)-program with 0-ary query relations maintains a regular language [10, Theorem 3.2]. ◀

To highlight the role of the Sunflower Lemma for the proof of Theorem 10 (b), we give a full exposition of this proof in the following. At first, we sketch the basic proof idea for consistent DynProp(1-aux)-programs over graphs, i.e., the input schema contains a single binary relation symbol E. For simplicity we also assume a 0-ary query relation. The general statement requires more machinery and is proved below.

Our goal is to show that if such a program $\mathcal{P}$ accepts some graph then it also accepts one with "few" edges, where "few" only depends on the schema of the program. To this end we show that if a graph G accepted by $\mathcal{P}$ contains many edges then one can find a large "well-behaved" edge set in G from which edges can be removed without changing the result of $\mathcal{P}$. Emptiness can then be tested in a brute-force manner by trying out insertion sequences for all graphs with few edges (over a canonical domain $\{1, \ldots, n\}$).

More concretely, we consider an edge set "well-behaved", if it consists only of self-loops, it is a set of disjoint non-self-loop-edges, or is is a *star*, that is, the edges share the same source node or the same target node. From the Sunflower Lemma [9] it follows that for every $p \in \mathbb{N}$ there is an $N_p \in \mathbb{N}$ such that every (directed) graph with N_p edges contains p self-loops, or p disjoint edges, or a star with p edges.

Let us now assume, towards a contradiction, that the minimal graph accepted by $\mathcal{P}$ has N edges with $N > N_{M^2+1}$, where M is the number of binary (atomic) types over the schema $\tau = \tau_{\text{in}} \cup \tau_{\text{aux}}$ of $\mathcal{P}$. Then G either contains $M^2 + 1$ self-loops, or $M^2 + 1$ disjoint edges, or a $(M^2 + 1)$-star.

Let us assume first that G has a set $D \subseteq E$ of $M^2 + 1$ disjoint edges. We consider the state $\mathcal{S}$ reached by $\mathcal{P}$ after inserting all edges from $E \setminus D$ into the initially empty graph. Since D contains $M^2 + 1$ edges, there is a subset $D' \subseteq D$ of size $M + 1$ such that all edges in D' have the same atomic type in state $\mathcal{S}$. Let $\mathcal{S}_0$ be the state reached by $\mathcal{P}$ after inserting all edges in $D \setminus D'$ in $\mathcal{S}$. All edges in D' still have the same type in $\mathcal{S}_0$ since $\mathcal{P}$ is a quantifier-free program (though this type can differ from the type in $\mathcal{S}$). Let $e_1, \ldots, e_{M+1}$ be the edges in D' and denote by $\mathcal{S}_i$ the state reached by $\mathcal{P}$ after inserting $e_1, \ldots, e_i$ in $\mathcal{S}_0$. For each i, all edges $e_{i+1}, \ldots, e_{M+1}$ have the same type γ_i in state $\mathcal{S}_i$, again. As the number of binary atomic types is M, there are $i < j$ such that $\gamma_i = \gamma_j$, thus e_{M+1} has the same type in $\mathcal{S}_i$ and $\mathcal{S}_j$. Therefore, inserting the edges $e_{j+1}, \ldots, e_{M+1}$ in $\mathcal{S}_i$ yields a state with the same query bit as inserting those edges in $\mathcal{S}_j$. As the query bit in the latter case is accepting, it is also accepting in the former case, yet in that case the underlying graph has fewer edges than G, the desired contradiction. The case where G contains $M^2 + 1$ self-loops is completely analogous.

Now assume that G contains a star with $M^2 + 1$ edges. The argument is very similar to the argument for disjoint edges. First insert all edges not involved in the star into an initially empty graph. Then there is a set D of many star edges of the same type, and they still have

the same type after inserting the other edges of the star. A graph with fewer edges that is accepted by $\mathcal{P}$ can then be obtained as above.

The idea generalizes to input schemata with larger arity by applying the Sunflower Lemma in order to obtain a "well-behaved" sub-relation within an input relation that contains many tuples. In order to prove this generalization we first recall the Sunflower Lemma, and observe that it has an analogon for tuples.

The Sunflower Lemma was introduced in [9], here we follow the presentation in [14]. A *sunflower* with p *petals* and a *core* Y is a collection of p sets $S_1, \ldots, S_p$ such that $S_i \cap S_j = Y$ for all $i \neq j$.

▶ **Lemma 11** (Sunflower Lemma, [9]). *Let $p \in \mathbb{N}$ and let $\mathcal{F}$ be a family of sets each of cardinality ℓ. If $\mathcal{F}$ consists of more than $N_{\ell,p} \stackrel{def}{=} \ell!(p-1)^\ell$ sets then $\mathcal{F}$ contains a sunflower with p petals.*

We call a set H of tuples of some arity ℓ a *sunflower (of tuples)* if it has the following three properties.

(i) All tuples in H have the same equality type.
(ii) There is a set $J \subset \{1, \ldots, \ell\}$ such that $t_j = t'_j$ for every $j \in J$ and all tuples $t, t' \in H$.
(iii) For all tuples $t \neq t'$ in H the sets $\{t_i \mid i \notin J\}$ and $\{t'_i \mid i \notin J\}$ are disjoint.

We say that H has $|H|$ petals.

The following Sunflower Lemma for tuples has been stated in various variants in the literature, e.g., in [17, 15].

▶ **Lemma 12** (Sunflower Lemma for tuples). *Let $\ell, p \in \mathbb{N}$ and let R be a set of ℓ-tuples. If R contains more than $\bar{N}_{\ell,p} \stackrel{def}{=} \ell^\ell p^\ell (\ell!)^2$ tuples then it contains a sunflower with p petals.*

Proof. Let R be an ℓ-ary relation that contains $\bar{N}_{\ell,p}$ tuples. As there are less than ℓ^ℓ equality types of ℓ-tuples there is a set $R' \subseteq R$ of size at least $p^\ell(\ell!)^2$, in which all tuples have the same equality type. Application of Lemma 2 in [15] yields[11] a sunflower with p petals. ◀

It is instructive to see how Lemma 12 shows that a graph with sufficiently many edges has many selfloops, disjoint edges or a large star: Selfloops correspond to the equality type of tuples (t_1, t_2) with $t_1 = t_2$, many disjoint edges to the case $J = \emptyset$ and the two possible kinds of stars to $J = \{1\}$ and $J = \{2\}$, respectively.

Proof (of Theorem 10 (b)). Now the proof for binary input schemas easily translates to general input schemas. For the sake of completeness we give a full proof.

Suppose that a consistent DynProp(1-aux)-program $\mathcal{P}$ over schema τ with 0-ary[12] query relation accepts an input database $\mathcal{D}$ that contains at least one relation R with many tuples.

Suppose that R is of arity ℓ and contains $\bar{N}_{\ell,M^2+1}$ diverse tuples where M is the number of ℓ-ary (atomic) types over the schema of $\mathcal{P}$. We show that $\mathcal{P}$ already accepts a database with less tuples than $\mathcal{D}$.

By Lemma 12, R contains a sunflower R' of size M^2+1. Consider the state $\mathcal{S}$ reached by $\mathcal{P}$ after inserting all tuples from $R \setminus R'$ into the initially empty database. Since R' contains M^2+1 tuples, there is a subset $R'' \subseteq R'$ of size $M+1$ such that all tuples in R'' have the same atomic type in state $\mathcal{S}$. Let $\mathcal{S}_0$ be the state reached by $\mathcal{P}$ after inserting all tuples in

[11] In [15], elements from the "outer part" of a petal can also occur in the "core". As in R' all tuples have the same equality type, this can not happen in our setting.
[12] At the end of the proof we discuss how to deal with unary query relations.

$R' \setminus R''$ in $\mathcal{S}$. All tuples in R'' still have the same type in $\mathcal{S}_0$ since $\mathcal{P}$ is a quantifier-free program (though this type can differ from the type in $\mathcal{S}$).

Let $\vec{a}_1, \ldots, \vec{a}_{M+1}$ be the tuples in R'' and denote by $\mathcal{S}_i$ the state reached by $\mathcal{P}$ after inserting $a_1, \ldots, a_i$ in $\mathcal{S}_0$. In state $\mathcal{S}_i$ all tuples $a_{i+1}, \ldots, a_{M+1}$ have the same type, again. As the number of ℓ-ary atomic types is k, there are $i < j$ such that a_{M+1} has the same type in $\mathcal{S}_i$ and $\mathcal{S}_j$. Therefore, inserting the edges $e_{j+1}, \ldots, e_{M+1}$ in $\mathcal{S}_i$ yields a state with the same query bit as inserting this sequence in $\mathcal{S}_j$. As the query bit in the latter case is accepting, it is also accepting in the former case, yet in that case the underlying database has fewer tuples than $\mathcal{D}$, the desired contradiction.

If $\mathcal{P}$ has a unary query relation, then the proof has to be adapted as follows. For an accepted database $\mathcal{D}$, the unary query relation contains some element a. Now M is chosen as the number of $(\ell+1)$-ary atomic types (instead of the number of ℓ-ary atomic types), and R'' is chosen as sub-sunflower where all tuples $(\vec{a}_1, a), \ldots, (\vec{a}_{M+1}, a)$ have the same atomic type. The rest of the proof is analogous. ◀

The final result of this subsection gives a characterization of the class of queries maintainable by consistent DynProp(0-aux)-programs. This characterization is not needed to obtain decidability of the emptiness problem for such queries, since this is included in Theorem 10. However, we consider it interesting in its own right.

▶ **Theorem 13.** *A Boolean query $\mathcal{Q}$ can be maintained by a consistent* DynProp(0-*aux*) *program if and only if it is a modulo query.*

Intuitively[13], a *modulo query* is a Boolean combination of constraints of the form: the number of tuples that have some atomic type γ is q modulo p, for some natural numbers $p \geq 2$ and $q < p$.

4.3 The impact of built-in orders

A closer inspection of the proof that the emptiness problem is undecidable for consistent DynFO(1-in, 2-aux)-programs (Theorem 8) reveals that the construction only requires one binary auxiliary relation: a linear order on the "active" elements. The proof would also work if a global linear order on all elements of the domain would be given. We say that a dynamic program has a *built-in linear order*, if there is one auxiliary relation $R_<$ that is always initialized by a linear order on the domain and never changed. Likewise, for a *built-in successor relation.*

That is, the border of undecidability for consistent DynFO-programs actually lies between consistent DynFO(1-in, 1-aux)-programs and consistent DynFO(1-in, 1-aux)-programs with a built-in linear order. Similarly, the border of undecidability for (not necessarily consistent) DynProp-programs actually lies between DynProp(1-in, 1-aux)-programs and DynProp(1-in, 1-aux)-programs with a built-in linear order.

▶ **Proposition 14.** *The emptiness problem is undecidable for*

(a) *consistent* DynFO(1-*in*, 1-*aux*)*-programs with a built-in linear order or a built-in successor relation,*

(b) DynProp(1-*in*, 1-*aux*)*-programs with a built-in successor relation.*

However, for dynamic programs that only have auxiliary bits, linear orders or successor relations do not affect decidability.

[13] The actual formalization uses sets of elements rather than tuples.

▶ **Proposition 15.** *The emptiness problem is decidable for*

(a) *consistent* DynFO(1-*in*, 0-*aux*)*-programs with a built-in linear order or a built-in successor relation,*

(b) DynProp(1-*in*, 0-*aux*)*-programs with a built-in linear order or a built-in successor relation.*

5 The Consistency Problem

In Section 4.2 we studied Emptiness for classes of consistent dynamic programs. It turned out that with this restriction the emptiness problem is easier than for general dynamic programs. One might thus consider the following approach for testing whether a given general dynamic program is empty: Test whether the program is consistent and if it is, use an algorithm for consistent programs. To understand whether this approach can be helpful, we study the following algorithmic problem, parameterized by a class $\mathcal{C}$ of dynamic programs.

Problem:	Consistency($\mathcal{C}$)
Input:	A dynamic program $\mathcal{P} \in \mathcal{C}$ with FO initialization
Question:	Is $\mathcal{P}$ a consistent program with respect to empty initial databases?

It is not very surprising that Consistency is not easier than Emptiness, since deciding Emptiness boils down to finding *one* modification sequence resulting in a state with particular properties and Consistency is about finding *two* modification sequences resulting in two states with particular properties. This high level comparison can actually be turned into rather easy reductions from Emptiness to Consistency.

On the other hand, Consistency can also be reduced to Emptiness. For this direction the key idea is to simulate two modification sequences simultaneously and to integrate their resulting states into one joint state. This is easy if quantification is available, and requires some work for DynProp-fragments.

▶ **Theorem 16.** *Let* $\ell \geq 1, m \geq 0$.

(a) *For every* $\mathcal{C} \in \{\text{DynFO}(\ell\text{-}in, m\text{-}aux), \text{DynFO}(\ell\text{-}in), \text{DynFO}(m\text{-}aux), \text{DynFO}\}$,

  (i) $\text{Emptiness}(\mathcal{C}) \leq \text{Consistency}(\mathcal{C})$, *and*

  (ii) $\text{Consistency}(\mathcal{C}) \leq \text{Emptiness}(\mathcal{C})$.

(b) *For every*
$\mathcal{C} \in \{\text{DynProp}(\ell\text{-}in, m\text{-}aux), \text{DynProp}(\ell\text{-}in), \text{DynProp}(m\text{-}aux), \text{DynProp}\}$,

  (i) $\text{Emptiness}(\mathcal{C}) \leq \text{Consistency}(\mathcal{C})$, *and*

  (ii) $\text{Consistency}(\mathcal{C}) \leq \text{Emptiness}(\mathcal{C})$.

6 The History Independence problem

As discussed in Section 4.2, it is natural to expect that a dynamic program is consistent, i.e., that the query relation only depends on the current database, but not on the modification sequence by which it has been reached. Many dynamic programs in the literature satisfy a stronger property: not only their query relation but *all* their auxiliary relations depend only on the current database. Formally, we call a dynamic program *history independent* if all auxiliary relations in $\mathcal{P}_\alpha(\mathcal{D})$ only depend on $\alpha(\mathcal{D})$, for all modification sequences α and initial empty databases $\mathcal{D}$. History independent dynamic programs (also called *memoryless* [19] or *deterministic* [4]) are still expressive enough to maintain interesting queries like undirected reachability [11], but also some lower bounds are known for such programs [4, 11, 25].

In this section, we study decidability of the question whether a given dynamic program is history independent.

Problem:	HistoryIndependence($\mathcal{C}$)
Input:	A dynamic program $\mathcal{P} \in \mathcal{C}$ with FO initialization
Question:	Is $\mathcal{P}$ history independent with respect to empty initial databases?

Not surprisingly, HistoryIndependence is undecidable in general. This can be shown basically in the same way as the general undecidability of Emptiness in Theorem 5.

▶ **Theorem 17.** HistoryIndependence *is undecidable for* DynFO(2-*in*, 0-*aux*)*-programs.*

However, the precise borders between decidable and undecidable fragments are different for HistoryIndependence than for Emptiness and Emptiness for consistent programs. More precisely, HistoryIndependence is decidable for DynFO- and DynProp-programs with unary input databases, and for DynProp-programs with unary auxiliary databases.

For showing that HistoryIndependence is decidable for DynFO-programs with unary input databases, we prove that if such a program is not history independent then this is witnessed by some reachable small "bad state". A decision algorithm can then simply test whether such a state exists. Bad states satisfy one of two properties: they either locally contradict history independence or they are "inhomogeneous". We define both notions in the following.

A state $\mathcal{S}$ over domain D is *locally history independent*[14] for a dynamic program $\mathcal{P}$ if the following three conditions hold.

(H1) $\mathcal{P}_{\delta_1\delta_2}(\mathcal{S}) = \mathcal{P}_{\delta_2\delta_1}(\mathcal{S})$ for all insertions δ_1 and δ_2.
(H2) $\mathcal{S} = \mathcal{P}_{\text{INS}_R(\vec{a})\text{DEL}_R(\vec{a})}(\mathcal{S})$ if $\vec{a} \notin R^{\mathcal{S}}$, for all $R \in \tau_{\text{in}}$ and $\vec{a}$ over D.
(H3) $\mathcal{S} = \mathcal{P}_{\text{INS}_R(\vec{a})}(\mathcal{S})$ if $\vec{a} \in R^{\mathcal{S}}$ and $\mathcal{S} = \mathcal{P}_{\text{DEL}_R(\vec{a})}(\mathcal{S})$ if $\vec{a} \notin R^{\mathcal{S}}$, for all $R \in \tau_{\text{in}}$ and $\vec{a}$ over D.

Locally history independence is well-suited to algorithmic analysis. The following lemma shows that for testing history independence it actually suffices to test locally history independence for all states reachable by very restricted modification sequences.

▶ **Lemma 18.** *Let $\mathcal{P}$ be a dynamic program.*

(a) *$\mathcal{P}$ is history independent if and only if every state reachable by $\mathcal{P}$ via insertion sequences is locally history independent.*
(b) *If $\mathcal{P}$ is a* DynFO(1-*in*)*-program, then $\mathcal{P}$ is history independent if and only if every state reachable by $\mathcal{P}$ via insertion sequences in normal form is locally history independent.*

A state $\mathcal{S}$ is *homogeneous* if all tuples $\vec{a}$ and $\vec{b}$ with the same (atomic) τ_{in}-type also have the same (atomic) τ_{aux}-type. The following lemma is an immediate consequence of [4, Lemma 16].

▶ **Lemma 19.** *For every history independent* DynFO(1-*in*)*-program, every reachable state is homogeneous.*

We call a state of a DynFO(1-in)-program that is not homogeneous or not locally history independent a *bad state*. The following lemma is the key ingredient for deciding history independence for DynFO(1-in)-programs. It restricts the size of the smallest bad state and therefore allows for testing history independence in a brute-force manner.

[14] We define this term for arbitrary input arity, since the first part of Lemma 18 holds in general.

▶ **Proposition 20.** *Let $\mathcal{P}$ be a* DynFO(1-*in*, *m*-*aux*)*-program. There is a number $N \in \mathbb{N}$, that can be computed from $\mathcal{P}$, such that if $\mathcal{P}$ is* not *history independent, then there exists an empty database $\mathcal{D}_\emptyset$ of size at most N and an insertion sequence α in normal form such that $\mathcal{P}_\alpha(\mathcal{D}_\emptyset)$ is bad.*

▶ **Theorem 21.** HistoryIndependence *is decidable for* DynFO(1-*in*)*-programs.*

Using the same technique as in the proof of Theorem 10 (b), history independence can be shown to be decidable for DynProp(1-aux)-programs.

▶ **Theorem 22.** HistoryIndependence *is decidable for* DynProp(1-*aux*)*-programs.*

7 Conclusion

In this work we studied the algorithmic properties of static analysis problems for (restrictions of) dynamic programs. Most of the results are summarized in Table 1. In general only very strong restrictions yield decidability.

The only cases left open are about DynProp-programs when both the arity of the input and the arity of the auxiliary relations is at least 2. For such programs the status of history independence and emptiness of consistent remains open. We conjecture that for history independence the decidable fragment of DynProp is larger than exhibited here.

Our results will hopefully contribute to a better understanding of the power of dynamic programs. On the one hand the undecidability proofs show that very restricted dynamic programs can already simulate powerful machine models. It is natural to ask whether this power can be used to maintain other, more common queries. On the other hand the decidability results utilize limitations of the state space and the transition between states for classes of restricted programs. Such limitations can be a good starting point for the development of techniques for proving lower bounds for the respective fragments.

References

1 Samir Datta, Raghav Kulkarni, Anish Mukherjee, Thomas Schwentick, and Thomas Zeume. Reachability is in DynFO. In *ICALP*, pages 159–170, 2015.

2 Guozhu Dong, Leonid Libkin, and Limsoon Wong. On impossibility of decremental recomputation of recursive queries in relational calculus and SQL. In *DBPL*, page 7, 1995.

3 Guozhu Dong, Leonid Libkin, and Limsoon Wong. Incremental recomputation in local languages. *Inf. Comput.*, 181(2):88–98, 2003.

4 Guozhu Dong and Jianwen Su. Deterministic FOIES are strictly weaker. *Ann. Math. Artif. Intell.*, 19(1-2):127–146, 1997.

5 Guozhu Dong and Jianwen Su. Arity bounds in first-order incremental evaluation and definition of polynomial time database queries. *J. Comput. Syst. Sci.*, 57(3):289–308, 1998.

6 Guozhu Dong, Jianwen Su, and Rodney Topor. Nonrecursive incremental evaluation of datalog queries. *Annals of Mathematics and Artificial Intelligence*, 14, 1995.

7 Catherine Dufourd, Alain Finkel, and Philippe Schnoebelen. Reset nets between decidability and undecidability. In *ICALP*, pages 103–115, 1998.

8 Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite model theory*. Perspectives in Mathematical Logic. Springer, 1995.

9 Paul Erdős and Richard Rado. Intersection theorems for systems of sets. *Journal of the London Mathematical Society*, s1-35(1):85–90, 1960.

10 Wouter Gelade, Marcel Marquardt, and Thomas Schwentick. The dynamic complexity of formal languages. *ACM Trans. Comput. Log.*, 13(3):19, 2012.

11 Erich Grädel and Sebastian Siebertz. Dynamic definability. In *ICDT*, pages 236–248, 2012.

12 William Hesse. *Dynamic Computational Complexity*. PhD thesis, University of Massachusetts Amherst, 2003.

13 John E. Hopcroft and Jean-Jacques Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Theor. Comput. Sci.*, 8:135–159, 1979.

14 Stasys Jukna. *Extremal combinatorics*, volume 2. Springer, 2001.

15 Stefan Kratsch and Magnus Wahlström. Preprocessing of min ones problems: A dichotomy. In *ICALP*, pages 653–665, 2010.

16 Leonid Libkin. *Elements of Finite Model Theory*. Springer, 2004.

17 Dániel Marx. Parameterized complexity of constraint satisfaction problems. *Computational Complexity*, 14(2):153–183, 2005.

18 Marvin L. Minsky. *Computation: finite and infinite machines*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1967.

19 Sushant Patnaik and Neil Immerman. Dyn-FO: A parallel, dynamic complexity class. In *PODS*, pages 210–221. ACM Press, 1994.

20 Thomas Schwentick, Nils Vortmeier, and Thomas Zeume. Static analysis for logic-based dynamic programs. *CoRR*, abs/1507.04537, 2015.

21 Boris A. Trahtenbrot. Impossibility of an algorithm for the decision problem in finite classes. *AMS Translations, Series 2*, 23:1–5, 1963.

22 Nils Vortmeier. Komplexitätstheorie verlaufsunabhängiger dynamischer Programme. Master's thesis, TU Dortmund University, 2013. In German.

23 Thomas Zeume. The dynamic descriptive complexity of k-clique. In *MFCS*, pages 547–558, 2014.

24 Thomas Zeume and Thomas Schwentick. Dynamic conjunctive queries. In *ICDT*, pages 38–49, 2014.

25 Thomas Zeume and Thomas Schwentick. On the quantifier-free dynamic complexity of reachability. *Inf. Comput.*, 240:108–129, 2015.

Sub-classical Boolean Bunched Logics and the Meaning of Par

James Brotherston[1] and Jules Villard[2]

1 **Department of Computer Science, University College London, UK**
2 **Department of Computing, Imperial College London, UK**

Abstract

We investigate intermediate logics between the bunched logics Boolean BI and Classical BI, obtained by combining classical propositional logic with various flavours of Hyland and De Paiva's full intuitionistic linear logic. Thus, in addition to the usual multiplicative conjunction (with its adjoint implication and unit), our logics also feature a multiplicative *disjunction* (with its adjoint co-implication and unit). The multiplicatives behave "sub-classically", in that disjunction and conjunction are related by a *weak distribution* principle, rather than by De Morgan equivalence.

We formulate a Kripke semantics, covering all our sub-classical bunched logics, in which the multiplicatives are naturally read in terms of resource operations. Our main theoretical result is that validity according to this semantics coincides with provability in a corresponding Hilbert-style proof system.

Our logical investigation sheds considerable new light on how one can understand the multiplicative disjunction, better known as linear logic's "par", in terms of resource operations. In particular, and in contrast to the earlier Classical BI, the models of our logics include the heap-like memory models of separation logic, in which disjunction can be interpreted as a property of intersection operations over heaps.

1998 ACM Subject Classification F.3.1 Logics and Meanings of Programs, F.4.1 Mathematical Logic and Formal Languages

Keywords and phrases Bunched logic, linear logic, modal logic, Kripke semantics, model theory

Digital Object Identifier 10.4230/LIPIcs.CSL.2015.325

1 Introduction

Bunched logics, which are free combinations of a standard propositional logic with some variety of multiplicative linear logic [3, 19], have applications in computer science as a means of expressing and manipulating properties of *resource* [18, 20]. Most notably, *separation logic* [21], which has been successfully employed in large-scale program verification [7, 22, 14] is based upon the bunched logic *Boolean BI* (BBI) obtained by combining ordinary classical logic with *multiplicative intuitionistic linear logic* (MILL) [13].

BBI has a simple Kripke semantics under which a formula of BBI is read as a set of elements ("resources") in an underlying *model*, essentially a generalised commutative monoid. The classical connectives have their usual meanings, and the MILL connectives (called *multiplicative*) are given "resource composition" readings: A multiplicative conjunction of formulas $A * B$ denotes those elements which divide, via the monoid operation, into two elements satisfying A and B respectively; the unit $\top^*$ of $*$ denotes the set of units of the monoid; and an implication (or "magic wand") $A \mathrel{-\!\!*} B$ denotes those elements that, when extended with an element satisfying A, always yield an element satisfying B.

24th EACSL Annual Conference on Computer Science Logic (CSL 2015).
Editor: Stephan Kreutzer; pp. 325–342

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this paper, we set out to answer the following question: What is the right way of adding a multiplicative *disjunction* – a.k.a. linear logic's notoriously tricky "par" – to BBI? A first answer to this question came previously in the study of *Classical BI* (CBI) [4], given by extending classical logic with *classical* multiplicative linear logic, i.e., MLL rather than MILL. Similar to MLL, the multiplicative disjunction $\mathbin{⅋}$ in CBI is the De Morgan dual of $*$ with respect to the multiplicative negation $\sim$: we have $(A \mathbin{⅋} B \equiv \sim(\sim A * \sim B))$. However, this is not very semantically informative. Furthermore, the heap-like models of BBI employed in separation logic (see e.g. [10]) turn out not to be models of CBI. This naturally raises the question of whether there might be bunched logics between BBI and CBI permitting the interpretation of multiplicative disjunction in such models.

Here, we shed new light on multiplicative disjunction by investigating "sub-classical" versions of bunched logic, under the common name BiBBI, obtained by combining classical logic with Hyland and De Paiva's *full intuitionistic linear logic* (FILL) [16]. In FILL, the conjunction $*$ and disjunction $\mathbin{⅋}$ are related not by De Morgan equivalence, but rather by *weak distribution*, i.e.

$$A * (B \mathbin{⅋} C) \vdash (A * B) \mathbin{⅋} C,$$

which follows from De Morgan equivalence, but is not equivalent to it. The disjunction $\mathbin{⅋}$ can also be endowed with a unit $\bot^*$ and an adjoint *co-implication*, $\mathbin{\backslash^*}$ ("magic slash").

We define provability in BiBBI simply by combining suitable Hilbert systems for classical logic and for FILL; the resulting Hilbert system can equivalently be reformulated as a *display calculus* proof system with the cut-elimination property, cf. [1, 3]. Our main technical contribution in this paper is a suitable Kripke frame semantics for BiBBI in which validity of BiBBI-formulas exactly coincides with provability. We obtain completeness of provability for validity in our semantics by embedding BiBBI into a suitable modal logic and deploying Sahlqvist's well-known completeness theorem (see e.g. [2]).

We consider a number of variants of BiBBI, based on whether or not various natural logical principles of FILL are included. For each such principle, we can write down an equivalent first-order condition on the Kripke models of BiBBI, with the frame condition corresponding to the above weak distribution law being particularly interesting. This fact enables us to present soundness and completeness results that are modular with respect to any choice of BiBBI-variant from our considered class.

We also undertake an investigation into the models of BiBBI, and present some general constructions for building them. From the program logic perspective, perhaps the most interesting aspect of BiBBI is that the standard heap-like models of separation logic can be extended into BiBBI-models obeying the weak distribution law, by interpreting disjunction using a notion of *intersection* between heaps (and there are at least two natural such intersection operations). We show that the typical unit law for $\mathbin{⅋}$, given by $A \mathbin{⅋} \bot^* \equiv A$, must fail in such models. However, we also show how to build more complicated models in which both weak distribution and the unit law do hold.

The remainder of this paper is structured as follows. In Section 2 we recall the model-theoretic and proof-theoretic characterisations of BBI and CBI. We then introduce our sub-classical bunched logic BiBBI, via both a Kripke frame semantics and a Hilbert-style axiomatic proof system, in Section 3. In Section 4 we investigate the models of BiBBI in more detail, and present some general model constructions and conservativity results. Section 5 presents the details of our completeness proof, and Section 7 concludes.

Due to space limitations, the proofs of the results in this paper have been abbreviated. Most of the full proofs can be found in an associated technical report [5].

2 Boolean and Classical BI

In this section, we recall the basic characterisations of *provability* and *validity* (based on Kripke semantics) in the bunched logics BBI [17, 11] and CBI [4]. We assume a countably infinite set $\mathcal{V}$ of propositional variables, and write $\mathcal{P}(X)$ for the powerset of a set X.

2.1 Boolean BI

▶ **Definition 2.1.** *BBI-formulas* are built from propositional variables $P \in \mathcal{V}$ using the standard connectives $\top, \bot, \neg, \wedge, \vee, \to$ of propositional classical logic, and the so-called "multiplicative" connectives: the constant $\top^*$ and binary operators $*$ and $-\!\!*$. By convention, $\neg$ has the highest precedence, followed by $*$, $\wedge$ and $\vee$, with $\to$ and $-\!\!*$ having lowest precedence.

▶ **Definition 2.2.** *Provability* in BBI is given by extending a complete Hilbert system for classical logic with the following axioms and inference rules for $*$, $-\!\!*$ and $\top^*$. The "sequent" notation $A \vdash B$ is syntactic sugar for the formula $A \to B$.

$$A * (B * C) \vdash (A * B) * C \qquad A * B \vdash B * A \qquad A \vdash A * \top^* \qquad A * \top^* \vdash A$$

$$\frac{A_1 \vdash B_1 \quad A_2 \vdash B_2}{A_1 * A_2 \vdash B_1 * B_2} \qquad \frac{A * B \vdash C}{A \vdash B -\!\!* C} \qquad \frac{A \vdash B -\!\!* C}{A * B \vdash C}$$

▶ **Definition 2.3.** A *BBI-frame* is a a tuple $\langle W, \circ, E\rangle$, where W is a set (of "worlds"), $\circ : W \times W \to \mathcal{P}(W)$ and $E \subseteq W$. We extend $\circ$ pointwise to $\mathcal{P}(W) \times \mathcal{P}(W) \to \mathcal{P}(W)$ by $W_1 \circ W_2 = \bigcup_{w_1 \in W_1, w_2 \in W_2} w_1 \circ w_2$.

A BBI-frame $\langle W, \circ, E\rangle$ is a *BBI-model* if $\circ$ is commutative and associative, and $w \circ E = \{w\}$ for all $w \in W$. (By definition, the latter means that $\bigcup_{e \in E} w \circ e = \{w\}$ for all $w \in W$.) We call E the set of *units* of the model $\langle W, \circ, E\rangle$.

If in a BBI-model $M = \langle W, \circ, E\rangle$ we have $|w_1 \circ w_2| \leq 1$ for all $w_1, w_2 \in W$, then we say that M is *partial functional* and understand $\circ$ as a partial function of type $W \times W \rightharpoonup W$.

▶ **Example 2.4.** The standard *heap model* $\langle \text{Heaps}, \circ, \{e\}\rangle$ of separation logic [21] is defined as follows. First, $\text{Heaps} = \text{Loc} \rightharpoonup_{\text{fin}} \text{Val}$ is the set of partial functions mapping finitely many *locations* in Loc to *values* in Val (typically Loc, Val are both infinite sets, with $\text{Loc} \subset \text{Val}$). We write $\mathsf{dom}(h)$ for the set of locations on which h is defined. We define $h_1 \circ h_2$ to be the union of heaps h_1 and h_2 if $\mathsf{dom}(h_1)$ and $\mathsf{dom}(h_2)$ are disjoint (and undefined otherwise), and we let e be the *empty heap* with $\mathsf{dom}(e) = \emptyset$. It is straightforward to verify that $\langle \text{Heaps}, \circ, \{e\}\rangle$ is a partial functional BBI-model.

▶ **Definition 2.5.** Let $M = \langle W, \circ, E\rangle$ be a BBI-model. A *valuation* for M is a function $\rho : \mathcal{V} \to \mathcal{P}(W)$ assigning to each proposition P a set $\rho(P) \subseteq W$. Given a valuation ρ for M, a $w \in W$ and a BBI-formula A, we define the forcing relation $w \models_\rho A$ by induction on A:

$$\begin{array}{rcl}
w \models_\rho P & \Leftrightarrow & w \in \rho(P) \\
w \models_\rho \top & \Leftrightarrow & \text{always} \\
w \models_\rho \bot & \Leftrightarrow & \text{never} \\
w \models_\rho \neg A & \Leftrightarrow & w \not\models_\rho A \\
w \models_\rho A_1 \wedge A_2 & \Leftrightarrow & w \models_\rho A_1 \text{ and } w \models_\rho A_2 \\
w \models_\rho A_1 \vee A_2 & \Leftrightarrow & w \models_\rho A_1 \text{ or } w \models_\rho A_2 \\
w \models_\rho A_1 \to A_2 & \Leftrightarrow & w \models_\rho A_1 \text{ implies } w \models_\rho A_2 \\
w \models_\rho \top^* & \Leftrightarrow & w \in E \\
w \models_\rho A_1 * A_2 & \Leftrightarrow & \exists w_1, w_2 \in W.\ w \in w_1 \circ w_2 \text{ and } w_1 \models_\rho A_1 \text{ and } w_2 \models_\rho A_2 \\
w \models_\rho A_1 -\!\!* A_2 & \Leftrightarrow & \forall w', w'' \in W.\ \text{if } w'' \in w \circ w' \text{ and } w' \models_\rho A_1 \text{ then } w'' \models_\rho A_2
\end{array}$$

A is said to be *valid in* M if $w \models_\rho A$ for all valuations ρ and for all $w \in W$, and *BBI-valid* if it is valid in all BBI-models.

▶ **Theorem 2.6** ([11]). *A BBI-formula is BBI-valid if and only if it is BBI-provable.*

2.2 Classical BI

▶ **Definition 2.7.** *CBI-formulas* are defined as BBI-formulas (Defn. 2.1), except that they may also contain the "multiplicative falsum" constant $\bot^*$. We write $\sim A$ as an abbreviation for $A \mathrel{-\!\!*} \bot^*$, and $A \mathbin{⅋} B$ as an abbreviation for $\sim(\sim A * \sim B)$.

▶ **Definition 2.8.** Provability in CBI is defined as provability in the Hilbert system for BBI (Defn. 2.2) extended with the "double negation elimination" axiom, $\sim\sim A \vdash A$.

▶ **Definition 2.9.** A *CBI-model* is given by a tuple $\langle W, \circ, E, U\rangle$, where $\langle W, \circ, E\rangle$ is a BBI-model (see Defn. 2.3), $U \subseteq W$, and for each $w \in W$, there is a unique $-w \in W$ (the "dual" of w) satisfying $(w \circ -w) \cap U \neq \emptyset$.

Given a CBI-model $\langle W, \circ, E, U\rangle$, the condition in Defn. 2.9 induces a function $- : W \to W$ sending w to $-w$, and necessarily $--w = w$ for any $w \in W$ (see [4]). Moreover, extending $-$ pointwise to sets, it is easy to show that $-E = U$. Therefore, intuitively, $-$ should be understood as a sort of "inverse" function on worlds [4]. E.g., every Abelian group is trivially a CBI-model, with $-w$ the group inverse of w.

▶ **Definition 2.10.** A valuation for a CBI-model and satisfaction $w \models_\rho A$ of a CBI-formula A by the world w and valuation ρ are defined as for BBI (Defn. 2.5), except that we add the following clause for satisfaction of the multiplicative falsum: $w \models_\rho \bot^* \;\Leftrightarrow\; w \notin U$.

It is then straightforward to derive the following satisfaction clauses for $\sim$ and $⅋$:

$$\begin{array}{rcl} w \models_\rho \sim A & \Leftrightarrow & -w \not\models_\rho A \\ w \models_\rho A \mathbin{⅋} B & \Leftrightarrow & \forall w_1, w_2 \in W.\ \text{if } w \in -(-w_1 \circ -w_2) \text{ then } w_1 \models_\rho A \text{ or } w_2 \models_\rho B \end{array}$$

▶ **Theorem 2.11** ([4, 3]). *A CBI-formula is CBI-valid if and only if it is CBI-provable.*

Unfortunately, CBI cannot be used to reason about heap-like memory models:

▶ **Proposition 2.12.** *Given the heap model* $\langle \text{Heaps}, \circ, \{e\}\rangle$ *of BBI defined in Example 2.4, there is no set* $U \subseteq \text{Heaps}$ *such that* $\langle \text{Heaps}, \circ, \{e\}, U\rangle$ *is a CBI-model.*

Proof. Suppose for contradiction that such a U exists. By the remark following Defn. 2.9, we have $U = -\{e\} = \{-e\}$. Note that $-e \in \text{Heaps}$ and thus $\mathsf{dom}(-e)$ is finite. Let h be a heap with $\mathsf{dom}(h) \supset \mathsf{dom}(-e)$ (there are infinitely many such h). Then there exists a heap $-h$ such that $h \circ -h = -e$ by the CBI-axiom, but it is clear that there is no such heap. ◀

▶ **Theorem 2.13** ([4]). *CBI is not conservative over BBI, i.e., there are BBI-formulas that are CBI-valid but not BBI-valid.*

3 BiBBI: Sub-classical Boolean bunched logic

In this section we introduce our *sub-classical* Boolean bunched logic, BiBBI, which extends BBI with multiplicative disjunction $⅋$, together with its adjoint co-implication $\backslash^*$ ("magic slash") and the multiplicative falsum $\bot^*$. We adopt the "Bi" prefix in BiBBI to remind

ourselves that, like in FILL [16], we have two families of multiplicative connectives, $(*, -\!\!*, \top^*)$ and $(\mathbin{\mathbb{\forall}}, \mathbin{\backslash^*}, \bot^*)$, that are not however connected by De Morgan equivalences.

First, we present a basic characterisation of Kripke validity for BiBBI-formulas and an associated notion of basic provability. Then, we consider a range of variants of the basic logic obtained by adding various logical laws from FILL (see Figure 1), which we regard as a sort of "logical buffet" from which we can pick and choose the principles we wish to include. (Commutativity of $\mathbin{\mathbb{\forall}}$ is considered a basic principle for technical convenience: a non-commutative $\mathbin{\mathbb{\forall}}$ naturally leads to both $\mathbin{\backslash^*}$ and $\bot^*$ splitting into two connectives.)

Our choice of models and interpretation achieves several complementary objectives:

1. BiBBI extends BBI and, furthermore, when a suitable "classicality" axiom is added to BiBBI, it collapses into CBI (see Prop. 3.9). Thus, the variants of BiBBI can be seen as intermediate logics between BBI and CBI.
2. We interpret multiplicative disjunction $\mathbin{\mathbb{\forall}}$ in BiBBI as a natural dual of multiplicative conjunction $*$, in that $\mathbin{\mathbb{\forall}}$ can be read as a binary *box modality* in modal logic [2], while $*$ can be read as a binary *diamond modality*.
3. For each natural logical principle of FILL governing $\mathbin{\mathbb{\forall}}$, $\mathbin{\backslash^*}$ and $\bot^*$, one can write down an equivalent first-order condition on BiBBI-models (see Figure 1).
4. Finally, for *any* variant of BiBBI obtained by taking some combination of logical axioms from Figure 1, we achieve soundness and completeness for that variant with respect to the associated class of models.

▶ **Definition 3.1.** A *BiBBI-formula* is defined as a BBI-formula (Defn. 2.1), except that it may also contain the multiplicative constant $\bot^*$, and the binary multiplicative connectives $\mathbin{\backslash^*}$ and $\mathbin{\mathbb{\forall}}$. As in CBI, we write ${\sim}A$ as an abbreviation for $A -\!\!* \bot^*$.

▶ **Definition 3.2.** A *basic BiBBI-model* is a tuple of the form $\langle W, \circ, E, \triangledown, U\rangle$, where $\langle W, \circ, E\rangle$ is a BBI-model, $U \subseteq W$ and $\triangledown\colon W \times W \to \mathcal{P}(W)$ is commutative. We extend $\triangledown$ pointwise to sets in a similar manner to $\circ$: $W_1 \triangledown W_2 = \bigcup_{w_1 \in W_1, w_2 \in W_2} w_1 \triangledown w_2$.

A valuation for a basic BiBBI-model $M = \langle W, \circ, E, \triangledown, U\rangle$ is defined as in Defn. 2.5. Satisfaction $w \models_\rho A$ of a BiBBI-formula A by the valuation ρ and world w is given by extending the forcing relation in Defn. 2.5 as follows:

$$\begin{array}{rcl} w \models_\rho \bot^* & \Leftrightarrow & w \notin U \\ w \models_\rho A \mathbin{\mathbb{\forall}} B & \Leftrightarrow & \forall w_1, w_2 \in W.\ \text{if } w \in w_1 \triangledown w_2 \text{ then } w_1 \models_\rho A \text{ or } w_2 \models_\rho B \\ w \models_\rho A \mathbin{\backslash^*} B & \Leftrightarrow & \exists w', w'' \in W.\ w'' \in w' \triangledown w \text{ and } w'' \models_\rho A \text{ and } w' \not\models_\rho B \end{array}$$

Similarly to BBI and CBI (see Section 2), a BiBBI-formula A is *valid in* M if $w \models_\rho A$ for all $w \in W$ and valuations ρ, and *BiBBI-valid* if it is valid in all BiBBI-models.

Intuitively, the binary operation $\triangledown$ and set U in a BiBBI-model $\langle W, \circ, E, \triangledown, U\rangle$ are used to interpret the connectives $\mathbin{\mathbb{\forall}}$, $\mathbin{\backslash^*}$ and $\bot^*$ in a way analogous to the use of $\circ$ and E to interpret $*$, $-\!\!*$ and $\top^*$. However, the analogy is not necessarily exact since, depending on the variant of BiBBI we consider, $\triangledown$ and U may exhibit quite different properties to $\circ$ and E. (For example, $\triangledown$ might fail to be associative.)

We note that the connective $\mathbin{\backslash^*}$ was not present in the original formulation of FILL, although Clouston et al. [9] recently showed that its addition to FILL is conservative. Here, observe that $\mathbin{\backslash^*}$ is interpreted as the natural adjoint of $\mathbin{\mathbb{\forall}}$ in basic BiBBI-models.

Principle	Axiom $\mathcal{A}$	Frame condition $\mathcal{F}(\mathcal{A})$
Associativity	$A \mathbin{\mathring{\vee}} (B \mathbin{\mathring{\vee}} C) \vdash (A \mathbin{\mathring{\vee}} B) \mathbin{\mathring{\vee}} C$	$w_1 \triangledown (w_2 \triangledown w_3) = (w_1 \triangledown w_2) \triangledown w_3$
Unit weakening	$A \vdash A \mathbin{\mathring{\vee}} \bot^*$	$w \triangledown U \subseteq \{w\}$
Unit contraction	$A \mathbin{\mathring{\vee}} \bot^* \vdash A$	$w \in w \triangledown U$
Contraction	$A \mathbin{\mathring{\vee}} A \vdash A$	$w \in w \triangledown w$
Weak distribution	$A * (B \mathbin{\mathring{\vee}} C) \vdash (A * B) \mathbin{\mathring{\vee}} C$	if $(x_1 \circ x_2) \cap (y_1 \triangledown y_2) \neq \emptyset$ then $\exists w.\ y_1 \in x_1 \circ w$ and $x_2 \in w \triangledown y_2$
Classicality	$\sim\!\sim A \vdash A$	$\exists! -w.\ (w \circ -w) \cap U \neq \emptyset$

Figure 1 Optional axioms of BiBBI and the corresponding first-order frame conditions (we suppress outermost universal quantifiers over the model domain).

▶ **Definition 3.3.** *Provability for basic BiBBI* is given by extending the proof system for BBI (see Defn. 2.2) with the following axioms and inference rules:

Monotonicity:
$$\frac{A_1 \vdash B_1 \quad A_2 \vdash B_2}{A_1 \mathbin{\mathring{\vee}} A_2 \vdash B_1 \mathbin{\mathring{\vee}} B_2}$$

Residuation:
$$\frac{A \vdash B \mathbin{\mathring{\vee}} C}{A \mathbin{\backslash^*} B \vdash C} \qquad \frac{A \mathbin{\backslash^*} B \vdash C}{A \vdash B \mathbin{\mathring{\vee}} C}$$

Commutativity:
$$A \mathbin{\mathring{\vee}} B \vdash B \mathbin{\mathring{\vee}} A$$

▶ **Theorem 3.4.** *If a formula A is provable for basic BiBBI (Defn. 3.3) then it is valid in all basic BiBBI-models.*

Proof (sketch). By soundness for standard BBI (Theorem 2.6) it suffices to show that the axioms and rules in Defn. 3.3 preserve validity in any basic BiBBI-model. ◀

▶ **Definition 3.5.** A *variant* of BiBBI is obtained by adding, for any combination of "principles" from Figure 1, (a) the logical axiom $\mathcal{A}$ for that principle to the basic BiBBI proof system in Defn. 3.3, and (b) the frame condition $\mathcal{F}(\mathcal{A})$ for that principle as an additional condition on the basic BiBBI-models in Defn. 3.2.

We investigate the variants of BiBBI and their models more closely in Section 4. For now, we just show that the correspondences laid out in Figure 1 are exact.

▶ **Theorem 3.6.** *For each principle in Figure 1, the axiom $\mathcal{A}$ is valid in a basic BiBBI-model M if and only if M satisfies the corresponding frame condition $\mathcal{F}(\mathcal{A})$.*

Proof (sketch). Let $M = \langle W, \circ, E, \triangledown, U\rangle$ be a basic BiBBI-model. We distinguish a case for each principle from Figure 1. Here we just show the most interesting cases: weak distribution and classicality.

Weak distribution: ($\Leftarrow$) Assuming that the weak distribution frame condition holds in M, we have to show that $A * (B \mathbin{\mathring{\vee}} C) \vdash (A * B) \mathbin{\mathring{\vee}} C$ is valid in M. So, given $w \models_\rho A * (B \mathbin{\mathring{\vee}} C)$, we must show $w \models_\rho (A * B) \mathbin{\mathring{\vee}} C$. This means showing, assuming $w \in w_1 \triangledown w_2$, that $w_1 \models_\rho A * B$ or $w_2 \models_\rho C$. Since we have $w \models_\rho A * (B \mathbin{\mathring{\vee}} C)$, we have $w \in x_1 \circ x_2$ where $x_1 \models_\rho A$ and $x_2 \models_\rho B \mathbin{\mathring{\vee}} C$. Thus we have $(x_1 \circ x_2) \cap (w_1 \triangledown w_2) \neq \emptyset$, so by the weak distribution property there exists $y \in W$ such that $w_1 \in x_1 \circ y$ and $x_2 \in y \triangledown w_2$. Now, since $x_2 \in y \triangledown w_2$ and $x_2 \models_\rho B \mathbin{\mathring{\vee}} C$ we have $y \models_\rho B$ or $w_2 \models_\rho C$. If $w_2 \models_\rho C$, we are done. If not, we have $w_1 \in x_1 \circ y$ and $x_1 \models_\rho A$ and $y \models_\rho B$, i.e., $w_1 \models_\rho A * B$ as required.

($\Rightarrow$) Assuming that $A * (B \mathbin{\mathring{\vee}} C) \vdash (A * B) \mathbin{\mathring{\vee}} C$ is valid in M, we have to show that the weak distribution frame condition holds in M. That is, supposing $z \in (x_1 \circ x_2) \cap (y_1 \triangledown y_2)$,

we need a $w \in W$ such that $y_1 \in x_1 \circ w$ and $x_2 \in w \triangledown y_2$. Let A, B, C be propositional variables and define a valuation ρ for M by

$$\rho(A) = \{x_1\} \ , \qquad \rho(B) = \{w \in W \mid x_2 \in w \triangledown y_2\} \ , \quad \text{and} \quad \rho(C) = W \setminus \{y_2\} \ .$$

We claim that $x_2 \models_\rho B \mathbin{\dot{\vee}} C$. To see this, let $x_2 \in w_1 \triangledown w_2$. By construction of ρ, if $w_2 \not\models_\rho C$ then $w_2 = y_2$ and hence $w_1 \models_\rho B$. Thus either $w_1 \models_\rho B$ or $w_2 \models_\rho C$ as required. Now, since $z \in x_1 \circ x_2$, with $x_1 \models_\rho A$ and $x_2 \models_\rho B \mathbin{\dot{\vee}} C$ by the above, we obtain $z \models_\rho A * (B \mathbin{\dot{\vee}} C)$. Since the weak distribution axiom is valid in M, we get $z \models_\rho (A * B) \mathbin{\dot{\vee}} C$. Then, as $z \models_\rho (A * B) \mathbin{\dot{\vee}} C$ and $z \in y_1 \triangledown y_2$ but $y_2 \not\models_\rho C$, we must have $y_1 \models_\rho A * B$. This means that there exist $u, w \in W$ with $y_1 \in u \circ w$ and $u \models_\rho A$ and $w \models_\rho B$. By definition of ρ, this means that $y_1 \in x_1 \circ w$ and $x_2 \in w \triangledown y_2$, as required.

Classicality: ($\Leftarrow$) Assuming the classicality condition, i.e. the CBI-model axiom, holds in M, we have to show that $\sim\!\sim A \vdash A$ is valid. Assume that $w \models_\rho \sim\!\sim A$. Using the clause for satisfaction of $\sim$ given in Section 2, we have $--w \models_\rho A$, and thus immediately $w \models_\rho A$ as required, using the fact (also from Section 2) that $--w = w$.

($\Rightarrow$) Assuming that $\sim\!\sim A \vdash A$ is valid in M, we have to show that, for any $w \in W$, there is a unique $w' \in W$ such that $(w \circ w') \cap U \neq \emptyset$. Let A be a propositional variable and define a valuation ρ for M by $\rho(A) = W \setminus \{w\}$. By construction, $w \not\models_\rho A$, so using the main assumption we have $w \not\models_\rho (A \mathrel{-\!\!*} \bot^*) \mathrel{-\!\!*} \bot^*$. Thus, there exist $w', w'' \in W$ such that $w'' \in w \circ w'$ and $w' \models_\rho A \mathrel{-\!\!*} \bot^*$ but $w'' \not\models_\rho \bot^*$, i.e. $w'' \in U$. That is, there exists an $-w = w' \in W$ such that $(w \circ -w) \cap U \neq \emptyset$.
It just remains to show that $-w$ is unique. Write $\mathrm{Co}(w)$ for the set of all w' such that $(w \circ w') \cap U \neq \emptyset$, and extend Co pointwise to sets as usual. Note that, by the above, $\mathrm{Co}(w)$ is nonempty. First we show that $\mathrm{Co}(\mathrm{Co}(w)) \subseteq \{w\}$. Define a new valuation ρ' for M by $\rho'(A) = \{w\}$, so that $w \models_{\rho'} A$ by construction. Since $A \vdash \sim\!\sim A$ is already provable in BBI, we have $w \models_{\rho'} (A \mathrel{-\!\!*} \bot^*) \mathrel{-\!\!*} \bot^*$. It is easy to show that this means that $w' \models_{\rho'} A$ for all $w' \in \mathrm{Co}(\mathrm{Co}(w))$, i.e., $\mathrm{Co}(\mathrm{Co}(w)) \subseteq \{w\}$ as required. Furthermore, letting $-w \in \mathrm{Co}(w)$, we have $(w \circ -w) \cap U \neq \emptyset$ and hence $(-w \circ w) \cap U \neq \emptyset$, i.e., $w \in \mathrm{Co}(\mathrm{Co}(w))$. Hence $\mathrm{Co}(\mathrm{Co}(w)) = \{w\}$. It is easy to see that $\mathrm{Co}(w)$ must then be a singleton set: if $w_1, w_2 \in \mathrm{Co}(w)$ then $\mathrm{Co}(w_1), \mathrm{Co}(w_2) \subseteq \mathrm{Co}(\mathrm{Co}(w)) = \{w\}$. Hence $\mathrm{Co}(w_1) = \mathrm{Co}(w_2) = \{w\}$, and so $\mathrm{Co}(\mathrm{Co}(w_1)) = \mathrm{Co}(\mathrm{Co}(w_2))$, i.e. $w_1 = w_2$ as required. This completes the proof. ◀

▶ **Corollary 3.7** (Soundness). *If a formula is provable in some variant of BiBBI then it is valid in that variant.*

Proof. Follows immediately from Theorems 3.4 and 3.6. ◀

We also have the converse completeness result:

▶ **Theorem 3.8** (Completeness). *If a BiBBI-formula is valid in some variant of BiBBI then it is provable in that variant.*

We defer the detailed proof of Theorem 3.8 until Section 5.

Turning to proof theory, we can reformulate the family of Hilbert-style proof systems above for BiBBI and its variants as a *display calculus* having the cut-elimination property, where each variant property in Figure 1 is captured by an optional structural rule in the calculus. We present our display calculus for BiBBI in Section 6.

To conclude this section, we show that CBI can be seen as the variant of BiBBI obeying the "classicality" axiom in Figure 1.

▶ **Proposition 3.9.** *BiBBI and CBI are related by the following:*

1. *For any BiBBI-model $\langle W, \circ, E, \triangledown, U\rangle$ satisfying the classicality axiom, the tuple $\langle W, \circ, E, U\rangle$ is a CBI-model.*
2. *If $\langle W, \circ, E, U\rangle$ is a CBI-model and we define $w_1 \triangledown w_2 = -(-w_1 \circ -w_2)$, then the tuple $\langle W, \circ, E, \triangledown, U\rangle$ is a BiBBI-model satisfying all axioms but contraction in Figure 1.*
3. *When CBI-models are identified with BiBBI-models as above, CBI-validity coincides with validity in the variant of BiBBI satisfying all properties but contraction in Figure 1.*

Proof. Part 1 of the proposition is immediate by construction. For part 2, let $\langle W, \circ, E, U\rangle$ be a CBI-model. It is immediate that $\langle W, \circ, E, \triangledown, U\rangle$ is a basic BiBBI-model. We have to check that $\langle W, \circ, E, \triangledown, U\rangle$ satisfies the required frame conditions. Classicality is exactly the CBI-model axiom, so is trivially satisfied (and consequently we have $--w = w$ for any $w \in W$ and $-E = U$). For associativity, we check:

$$\begin{array}{rcll} w_1 \triangledown (w_2 \triangledown w_3) & = & -(-w_1 \circ --(-w_2 \circ -w_3)) & \\ & = & -(-w_1 \circ (-w_2 \circ -w_3)) & \text{(since } --X = X\text{)} \\ & = & -((-w_1 \circ -w_2) \circ -w_3) & \text{(by associativity of } \circ\text{)} \\ & = & -(--(-w_1 \circ -w_2) \circ -w_3) & \text{(since } --X = X\text{)} \\ & = & (w_1 \triangledown w_2) \triangledown w_3 & \end{array}$$

For the unit axioms, we can similarly check that $U \triangledown w = \{w\}$. Finally, we must verify the weak distribution condition. Suppose $(x_1 \circ x_2) \cap (y_1 \triangledown y_2) \neq \emptyset$. That is, for some $z \in x_1 \circ x_2$ we have $z \in -(-y_1 \circ -y_2)$, i.e. $-z \in -y_1 \circ -y_2$, which is again equivalent (see [4]) to $y_1 \in z \circ -y_2$. Putting everything together and using associativity of $\circ$, we get $y_1 \in x_1 \circ (x_2 \circ -y_2)$. Thus, for some $w \in x_2 \circ -y_2$, we have $y_1 \in x_1 \circ w$. But, using the same properties as before, $w \in x_2 \circ -y_2$ is equivalent to $-x_2 \in -w \circ -y_2$ and then to $x_2 \in -(-w \circ -y_2)$, i.e. $x_2 \in w \triangledown y_2$ as required. This completes the verification.

Finally, for part 3, just observe that the clauses for satisfaction of $\bot^*$ coincide in the forcing relations for BiBBI and CBI, and that by inserting the definition of $\triangledown$ into BiBBI's clause for $\mathbin{⅋}$, we obtain exactly the usual CBI clause for $\mathbin{⅋}$. ◀

4 General constructions for BiBBI-models

In this section, we investigate the models of our variants of BiBBI, and present some general constructions for BiBBI-models, chiefly based on the heap-like models of BBI.

We begin with some simple constructions yielding conservativity results. Let $\langle W, \circ, E\rangle$ be a BBI-model. First, define $w \triangledown_= w' = \{w\}$ if $w = w'$, and $w \triangledown_= w' = \emptyset$ otherwise. Then $\langle W, \circ, E, \triangledown_=, W\rangle$ is easily seen to be a BiBBI-model satisfying associativity, unit weakening, unit contraction and contraction. Second, defining $w \triangledown_0 w' \stackrel{\text{def}}{=} \emptyset$ for all $w, w' \in W$, we have that $\langle W, \circ, E, \triangledown_0, U\rangle$ (for any $U \subseteq W$) is a BiBBI-model satisfying associativity, unit weakening and weak distribution. Consequently, we have:

▶ **Proposition 4.1.** *The variants of BiBBI given by: (a) associativity, unit weakening, unit contraction and contraction; and (b) associativity, unit weakening and weak distribution, are both* conservative *over BBI. That is, any BBI-formula valid in one of these variants is also BBI-valid.*

However, neither of the previous model constructions is very satisfying. In the first type, taking $\triangledown$ to be $\triangledown_=$, $A \mathbin{⅋} B$ simply becomes $A \vee B$. Moreover, as weak distribution does not hold in general, the $(*, -\!\!*, \top^*)$ and $(⅋, \mathbin{\backslash^*}, \bot^*)$ fragments of the logic are essentially disjoint; we are inclined to regard the variants of BiBBI without weak distribution as being

less interesting. On the other hand, under the second construction with $\triangledown$ being $\triangledown_0$, weak distribution does hold (trivially), but $A \mathbin{\dot{\triangledown}} B$ collapses into $\top$, which is even less interesting!

An immediate question is therefore whether there are BiBBI-models with weak distribution in which $\triangledown$ has a non-trivial interpretation. Our interest here is strictly in *sub-classical* models, i.e. those in which classicality does not hold, since classical models fall under the rubric of CBI, in which $w_1 \triangledown w_2$ should be read as $-(-w_1 \circ -w_2)$, cf. Proposition 3.9. We explore this question, and related ones, in the next two subsections. A second question is whether conservativity extends to the other sub-classical variants of BiBBI (e.g. the variant with all sub-classical properties from Figure 1). Our next result suggests that this is unlikely.

▶ **Definition 4.2.** A partial functional BBI-model $\langle W, \circ, E\rangle$:

- is *cancellative* if $w \circ w_1 = w \circ w_2 \neq \emptyset$ implies $w_1 = w_2$;
- is *extensible* if for all $w \in W$ there exists a $w' \in W \setminus E$ such that $w \circ w'$ is defined;
- has *indivisible units* if $w_1 \circ w_2 \in E$ implies $w_1, w_2 \in E$.

Note that the heap model of Example 2.4 satisfies all three properties above, as does, e.g., the total monoid $\langle \mathbb{N}, +, \{0\}\rangle$.

▶ **Proposition 4.3.** *Let $\langle W, \circ, E\rangle$ be a partial functional BBI-model that is cancellative, extensible and has indivisible units, as in Defn. 4.2. There does not exist a BiBBI-model of the form $\langle W, \circ, E, \triangledown, U\rangle$ satisfying weak distribution, unit weakening and unit contraction.*

Proof. Suppose for contradiction that $\langle W, \circ, E, \triangledown, U\rangle$ does exist. By unit contraction, U must be nonempty, so let $u \in U$. By extensibility, there is a $y \notin E$ such that $y \circ u$ is defined. By unit contraction, there exists $u' \in U$ such that $y \circ u \in (y \circ u) \triangledown u'$. Thus, by the weak distribution law, there exists $v \in W$ such that $y \circ u = y \circ v$ and $u \in v \triangledown u'$. By cancellativity, we obtain $v = u$ and thus $u \in u \triangledown u'$. By unit weakening and commutativity of $\triangledown$, we obtain $\{u\} = u \triangledown u' \subseteq \{u'\}$, and thus $u = u'$.

Now, since $y \circ u \in (y \circ u) \triangledown u'$, using $u = u'$ and the commutativity of $\triangledown$, we have $y \circ u \in u \triangledown (y \circ u)$. Then, by the standard unit law for BBI, there exists $e \in E$ such that $(y \circ u) \circ e \in u \triangledown (y \circ u)$. Thus, by weak distribution, there exists $w \in W$ such that $u = (y \circ u) \circ w$. As e is a unit for $y \circ u$, it is also a unit for u, so we have $e \circ u = (y \circ w) \circ u$. Hence, by cancellativity, $y \circ w = e \in E$ and so by the indivisible units property we have $y \in E$. But we already know $y \notin E$, contradiction. ◀

Proposition 4.3 demonstrates that in the class of BBI-models given by Defn. 4.2, which includes many standard examples, we are forced to choose between weak distribution and (at least one direction of) the unit law $A \mathbin{\dot{\triangledown}} \bot^* \equiv A$ when extending to a BiBBI-model. A BBI-formula whose validity implies membership of this class would yield nonconservativity of the BiBBI fragment with both weak distribution and unit weakening / contraction. Unfortunately, we have not yet been able to find such a formula. (We remark that the combination of weak distribution and unit contraction is particularly interesting, as it yields a multiplicative analogue of the usual *disjunctive syllogism*: $A * (\sim A \mathbin{\dot{\triangledown}} B) \vdash B$.)

The next two subsections present general constructions extending (certain types of partial functional) BBI-models to BiBBI-models obeying the weak distribution law.

4.1 Intersection in BBI-models

Our first approach to constructing BiBBI-models from BBI-models is to interpret $\triangledown$ as an "intersection-like" operator on worlds. This construction yields BiBBI-models with the contraction and weak distribution properties, but in general no others (Proposition 4.7).

As a motivating example, there are two natural ways one could go about defining intersection in the heap model of Example 2.4, depending on how one deals with *incompatibility*:

▶ **Example 4.4** (Heap intersections). We define two binary intersection operations $\cap_1$ and $\cap_2$ on heaps by:

$$(h_1 \cap_1 h_2)(\ell) \stackrel{\text{def}}{=} \begin{cases} h_1(\ell) & \text{if } \ell \in \mathsf{dom}(h_1) \cap \mathsf{dom}(h_2) \text{ and } h_1(\ell) = h_2(\ell) \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$h_1 \cap_2 h_2 \stackrel{\text{def}}{=} \begin{cases} h_1 \cap_1 h_2 & \text{if } h_1(\ell) = h_2(\ell) \text{ for all } \ell \in \mathsf{dom}(h_1) \cap \mathsf{dom}(h_2) \\ \text{undefined} & \text{otherwise} \end{cases}$$

The first intersection silently discards incompatible *parts* of heaps, while the second intersection requires the heaps to be fully compatible. Consequently, $\cap_1$ is associative, while $\cap_2$ is not. We note that neither $\cap_1$ nor $\cap_2$ has a natural set of units $U \subseteq \text{Heaps}$, in the sense that $h \cap_i U = \{h\}$ for all heaps h.

▶ **Proposition 4.5.** *Let* $\langle \text{Heaps}, \circ, \{e\}\rangle$ *be the heap model of Example 2.4, and let* $\cap_1$ *and* $\cap_2$ *be the heap intersection operations defined in Example 4.4. Then, for any* $U \subseteq \text{Heaps}$*, both* $\langle \text{Heaps}, \circ, \{e\}, \cap_1, U\rangle$ *and* $\langle \text{Heaps}, \circ, \{e\}, \cap_2, U\rangle$ *are BiBBI-models satisfying contraction and weak distribution (and the first also satisfies associativity).*

Unit contraction or unit weakening can easily be obtained in the above models by suitable choices of U, but, according to Prop. 4.3, it is impossible to obtain both simultaneously.

From now on, to simplify notation, and because most models of separation logic in the literature satisfy this constraint, we treat only partial functional BBI-models. Using associativity of $\circ$, we write $w_1 \sharp \ldots \sharp w_n$ to mean that $w_1 \circ \ldots \circ w_n$ is defined (i.e., non-empty). Then, we can extend Proposition 4.5 to arbitrary partial functional BBI-models, using a generalised version of the heap intersection $\cap_2$.

▶ **Definition 4.6.** Let $\langle W, \circ, E\rangle$ be a partial functional BBI-model, and define the operation $\triangledown_\cap \colon W \times W \to \mathcal{P}(W)$ by

$$w_1 \mathbin{\triangledown_\cap} w_2 = \{x \mid \exists x_1, x_2 \in W.\ w_1 = x \circ x_1 \text{ and } w_2 = x \circ x_2 \text{ and } x \sharp x_1 \sharp x_2\}.$$

In the heap model, $h_1 \triangledown_\cap h_2$ is exactly $h_1 \cap_2 h_2$, while in $\langle \mathbb{N}, +, \{0\}\rangle$ we have $n \triangledown_\cap m = \{k \mid n, m \geq k\}$. Note that $\triangledown_\cap$ is neither a partial function nor associative, in general.

▶ **Proposition 4.7.** *For any partial functional BBI-model* $M = \langle W, \circ, E\rangle$*, and any* $U \subseteq W$*, we have that* $\langle W, \circ, E, \triangledown_\cap, U\rangle$ *is a BiBBI-model satisfying contraction and weak distribution.*

Proof. Since M is a BBI-model and $\triangledown_\cap$ is commutative by construction, $\langle W, \circ, E, \triangledown_\cap, U\rangle$ is a basic BiBBI-model. To check contraction, let $w \in W$; we must show that $w \in w \mathbin{\triangledown_\cap} w$. This follows from the fact that, since M is a BBI-model, there is an $e \in E$ such that $w \circ e = w$, and thus $w \sharp e \sharp e$.

It remains to verify the weak distribution condition. That is, assuming $(x_1 \circ x_2) \cap (y_1 \mathbin{\triangledown_\cap} y_2) \neq \emptyset$, we require to find $w \in W$ such that $y_1 = x_1 \circ w$ and $x_2 \in w \mathbin{\triangledown_\cap} y_2$. By assumption, we have $x_1 \circ x_2 \in y_1 \mathbin{\triangledown_\cap} y_2$. By definition of $\triangledown_\cap$ there are z_1 and z_2 such that $y_1 = x_1 \circ x_2 \circ z_1$ and $y_2 = x_1 \circ x_2 \circ z_2$ and $(x_1 \circ x_2) \sharp z_1 \sharp z_2$. Now, letting $w = x_2 \circ z_1$, we immediately have $y_1 = x_1 \circ w$. To see that $x_2 \in w \mathbin{\triangledown_\cap} y_2$, we need $x', x'' \in W$ such that $w = x_2 \circ x'$ and $y_2 = x_2 \circ x''$ and $x_2 \sharp x' \sharp x''$. Choosing $x' = z_1$ and $x'' = x_1 \circ z_2$, we immediately have $w = x_2 \circ z_1$, and $y_2 = x_2 \circ x_1 \circ z_2$ by associativity. Finally, we must check $x_2 \sharp z_1 \sharp (x_1 \circ z_2)$, which follows by associativity from $(x_1 \circ x_2) \sharp z_1 \sharp z_2$. ◀

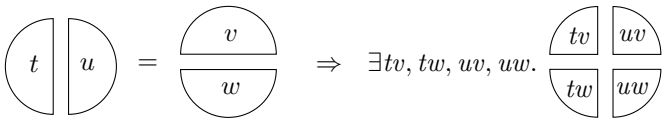
4.2 Intersection in BBI-models with environments

We now define our second general construction, based upon the one in the previous section, for constructing BiBBI-models obeying weak distribution, associativity, contraction *and* both unit laws. We require that the underlying BBI-model obeys the *cross-split* and *disjointness* properties typically encountered in heap-like models of separation logic [10, 6]:

▶ **Definition 4.8.** A partial functional BBI-model $M = \langle W, \circ, E\rangle$ has the *cross-split property* if for any $t, u, v, w \in W$ such that $t \circ u = v \circ w$, there exist tv, tw, uv, uw such that

$$t = tv \circ tw,\ u = uv \circ uw,\ v = tv \circ uv,\ \text{and}\ w = tw \circ uw.$$

Diagrammatically, this can be thought of in the following way:



M has the *disjointness* property if $w \sharp w$ implies $w \in E$.

We remark that, again, the standard heap model of Example 2.4 has both the cross-split and the disjointness property. The monoid $(\mathbb{N}, +, \{0\})$ does not satisfy disjointness (because $+$ is a total operation), but it does have the cross split property: Given $t + u = v + w$, simply take $tv = \min(t, v)$, $uw = \min(u, w)$, $tw = t - tv$ and $uv = u - uw$.

Given a BBI-model with the above properties, we construct a BiBBI-model $\bar{M} = \langle \bar{W}, \bar{\circ}, \bar{E}, \bar{\nabla}, D\rangle$, where each world in $\bar{W}$ consists of a "local" world $w \in W$ paired with a larger "environment" $x \in W$ such that $x = w \circ w'$ for some w'. On the "local" part of each world, $\bar{\circ}$ and $\bar{\nabla}$ behave as $\circ$ and $\nabla_\cap$, respectively. On the "environment" part of each world, $\bar{\circ}$ and $\bar{\nabla}$ behave as a *union* operation $\cup$ (as defined below) and the identity, respectively.

▶ **Definition 4.9.** Given a partial functional BBI-model $\langle W, \circ, E\rangle$, we define the *union* operation, $\cup : W \times W \to \mathcal{P}(W)$, by

$$w_1 \cup w_2 = \{y \circ y_1 \circ y_2 \mid w_1 = y \circ y_1 \text{ and } w_2 = y \circ y_2\}\ .$$

We lift $\cup$ to $\mathcal{P}(W) \times \mathcal{P}(W) \to \mathcal{P}(W)$ in the usual way: $W_1 \cup W_2 = \bigcup_{w_1 \in W_1, w_2 \in W_2} w_1 \cup w_2$.

For our purposes we shall require $\cup$ to be associative, which is not necessarily the case for arbitrary partial functional BBI-models.

▶ **Lemma 4.10.** *If a partial functional BBI-model $\langle W, \circ, E\rangle$ has the cross-split property, then $\cup$ in Definition 4.9 is associative. Moreover, if $w = w_1 \circ w_2$, then $w \in w \cup w_1$.*

▶ **Definition 4.11.** Let $M = \langle W, \circ, E\rangle$ be a partial functional BBI-model. We define $\bar{M} = \langle \bar{W}, \bar{\circ}, \bar{E}, \bar{\nabla}, D\rangle$ as follows:

$$\bar{W} = \{(w, x) \mid \exists w'.\ x = w \circ w'\} \qquad \bar{E} = \{(e, e) \mid e \in E\}$$

$$(w, x)\ \bar{\circ}\ (w', x') = \{(w \circ w', x'') \mid x'' \in x \cup x'\} \qquad D = \{(w, w) \mid w \in W\}$$

$$(w, x)\ \bar{\nabla}\ (w', x') = \begin{cases} \{(w'', x) \mid w'' \in w\ \nabla_\cap\ w'\} & \text{if } x = x' \\ \emptyset & \text{otherwise} \end{cases}$$

Instantiating M in the above definition with the heap model of Example 2.4, the set $\bar{W}$ pairs every heap with a larger heap that extends it, which can be thought of as pairing a local part of memory "owned" by a program with an "environment" reflecting the wider machine state.

Our main result about $\bar{M}$, stated as Theorem 4.13, is that, if M has the cross-split and the disjointness properties, then $\bar{M}$ is a BiBBI-model satisfying *all* the properties of Figure 1 (except classicality). The following lemma groups together a number of intermediary results used in the proof of this theorem.

▶ **Lemma 4.12.** *Suppose that $M = \langle W, \circ, E\rangle$ is partial functional and has the cross-split and disjointness properties, and let $\bar{M} = \langle \bar{W}, \bar{\circ}, \bar{E}, \bar{\nabla}, D\rangle$ be as in Definition 4.11. All of the following hold:*

1. *For all $(w_1, x), (w_2, x) \in \bar{W}$, we have $w_1 \nabla_\cap w_2$ a singleton set (and we typically drop the set brackets). Consequently, $\bar{\nabla}$ is a partial function on $\bar{W} \times \bar{W}$.*
2. *If $(w, x), (w_1 \circ w_2, x) \in \bar{W}$ with $w \sharp w_1$ and $w \sharp w_2$, then $(w \circ w_1 \circ w_2, x) \in \bar{W}$.*
3. *For all $(w, x), (w_1 \circ w_2, x) \in \bar{W}$, we have $w \nabla_\cap (w_1 \circ w_2) = (w \nabla_\cap w_1) \circ (w \nabla_\cap w_2)$.*

Proof (sketch). Each part of the lemma is proved directly; the proofs rely heavily on the disjointness and cross-split properties of M. ◀

▶ **Theorem 4.13.** *Given a partial functional BBI-model M with the cross-split and disjointness properties, $\bar{M}$ is a BiBBI-model with all the properties of Figure 1 except classicality.*

Proof (sketch). We check that $\bar{M}$ satisfies all properties of basic BiBBI-models, and all relevant properties from Figure 1, of which the most difficult case is, interestingly enough, the associativity of $\bar{\nabla}$. The verifications rely heavily on Lemmas 4.10 and 4.12. ◀

5 Completeness of BiBBI

This section presents our proof of completeness for (variants of) BiBBI, stated earlier as Theorem 3.8. Our approach follows the basic pattern previously employed in the literature for BBI [8] and for CBI [4]: we translate a given variant of BiBBI into modal logic, and appeal to Sahlqvist's well known completeness result (see e.g. [2]). Here, unsurprisingly, the weak distribution law of BiBBI presents the greatest technical obstacles to this approach.

We begin by recalling the standard definitions, from [2], of validity and provability in normal modal logic over a suitably chosen signature (a.k.a. "modal similarity type").

▶ **Definition 5.1.** A *modal logic formula* is built from propositional variables using the classical connectives, 0-ary modalities $\top^*$ and $\mathbf{U}$, and binary modalities $*, \mathrel{-\!\!\circ}, \nabla$ and $\mathrel{\leftarrow\!\!-}$.

▶ **Definition 5.2.** A *modal frame* is given by a tuple of the form $\langle W, \circ, \mathrel{-\!\!\circ}, \nabla, \mathrel{\leftarrow\!\!-}, E, U\rangle$, where $\circ$, $\mathrel{-\!\!\circ}$, ∇, and $\mathrel{\leftarrow\!\!-}$ all have type $W \times W \to \mathcal{P}(W)$, and $E, U \subseteq W$.

A valuation for a modal frame $M = \langle W, \ldots\rangle$ is as usual given by a function $\rho : \mathcal{V} \to \mathcal{P}(W)$. The forcing relation $w \models_\rho A$ is defined by induction on A in the standard way in modal logic, i.e. as for BBI in the case of propositional variables and classical connectives, with the following clauses for the modalities:

$$
\begin{array}{rcl}
w \models_\rho \top^* & \Leftrightarrow & w \in E \\
w \models_\rho \mathbf{U} & \Leftrightarrow & w \in U \\
w \models_\rho A * B & \Leftrightarrow & \exists w_1, w_2 \in W.\ w \in w_1 \circ w_2 \text{ and } w_1 \models_\rho A \text{ and } w_2 \models_\rho B \\
w \models_\rho A \mathrel{-\!\!\circ} B & \Leftrightarrow & \exists w_1, w_2 \in W.\ w \in w_1 \mathrel{-\!\!\circ} w_2 \text{ and } w_1 \models_\rho A \text{ and } w_2 \models_\rho B \\
w \models_\rho A \nabla B & \Leftrightarrow & \exists w_1, w_2 \in W.\ w \in w_1 \nabla w_2 \text{ and } w_1 \models_\rho A \text{ and } w_2 \models_\rho B \\
w \models_\rho A \mathrel{\leftarrow\!\!-} B & \Leftrightarrow & \exists w_1, w_2 \in W.\ w \in w_1 \mathrel{\leftarrow\!\!-} w_2 \text{ and } w_1 \models_\rho A \text{ and } w_2 \models_\rho B
\end{array}
$$

As usual, A is *valid* in M iff we have $w \models_\rho A$ for all $w \in W$ and valuations ρ.

Each of the binary functions $\circ, \multimap, \triangledown, \mathrel{-\!\!\blacktriangleleft}\colon W \times W \to \mathcal{P}(W)$ in a modal frame can be equivalently seen as a ternary relation over W (as is standard in modal logic). The corresponding modalities are each interpreted as a standard binary "diamond" modality.

▶ **Definition 5.3.** The *normal modal logic* $\mathbf{ML}_{\mathrm{BiBBI}}$ for the signature $(\top^*, \mathbf{U}, *, \multimap, \triangledown, \mathrel{-\!\!\blacktriangleleft})$ is given by extending a standard Hilbert system for classical logic with the following axioms and rules, for all $\otimes \in \{*, \multimap, \triangledown, \mathrel{-\!\!\blacktriangleleft}\}$:

$$\bot \otimes A \vdash \bot \text{ and } A \otimes \bot \vdash \bot$$
$$(A \vee B) \otimes C \vdash (A \otimes C) \vee (B \otimes C)$$
$$A \otimes (B \vee C) \vdash (A \otimes B) \vee (A \otimes C)$$

$$\frac{A_1 \vdash A_2 \quad B_1 \vdash B_2}{A_1 \otimes B_1 \vdash A_2 \otimes B_2}$$

Next, we recall the Sahlqvist completeness result for normal modal logics augmented with suitably well-behaved axioms, called *Sahlqvist formulas*. In fact, we only require so-called "very simple" Sahlqvist formulas for our completeness result.

▶ **Definition 5.4.** A *very simple Sahlqvist antecedent* (over the signature $(\top^*, \mathbf{U}, *, \multimap, \triangledown, \mathrel{-\!\!\blacktriangleleft})$) is given by the grammar: $S ::= P \mid \top \mid \bot \mid S \wedge S \mid \top^* \mid \mathbf{U} \mid S * S \mid S \multimap S \mid S \triangledown S \mid S \mathrel{-\!\!\blacktriangleleft} S$. A *very simple Sahlqvist formula* is an implication $A \vdash B$, where A is a very simple Sahlqvist antecedent and B is a *positive* modal logic formula (i.e., every propositional variable occurs within the scope of an even number of negations).

▶ **Theorem 5.5** (Sahlqvist [2])**.** *If a modal logic formula is valid in the set of all modal frames satisfying a set $\mathcal{A}$ of very simple Sahlqvist formulas, then it is provable in $\mathbf{ML}_{BiBBI} + \mathcal{A}$.*

We now define a set of Sahlqvist formulas that collectively capture all variants of BiBBI.

▶ **Definition 5.6.** For a given variant of BiBBI, define the set $\mathcal{A}_{\mathrm{BiBBI}}$ of very simple Sahlqvist formulas as follows:

(1)	$A \wedge (B * C) \vdash (B \wedge (C \multimap A)) * \top$	(Assoc.)	$A \triangledown (B \triangledown C) \vdash (A \triangledown B) \triangledown C$
(2)	$A \wedge (B \multimap C) \vdash \top \multimap (C \wedge (A * B))$	(Unit weak.)	$A \triangledown \mathbf{U} \vdash A$
(3)	$A \wedge (B \triangledown C) \vdash \top \triangledown (C \wedge (A \mathrel{-\!\!\blacktriangleleft} B))$	(Unit contr.)	$A \vdash A \triangledown \mathbf{U}$
(4)	$A \wedge (B \mathrel{-\!\!\blacktriangleleft} C) \vdash (B \wedge (C \triangledown A)) \mathrel{-\!\!\blacktriangleleft} \top$	(Contr.)	$A \vdash A \triangledown A$
(5)	$A * B \vdash B * A$	(Weak distr.)	$(A * B) \wedge (C \triangledown D) \vdash$
(6)	$A \triangledown B \vdash B \triangledown A$		$(A \wedge ((B \mathrel{-\!\!\blacktriangleleft} D) \multimap C)) * \top$
(7)	$A * (B * C) \vdash (A * B) * C$	(Classicality)	$(A \multimap \mathbf{U}) \multimap \mathbf{U} \vdash A$ and
(8)	$A * \top^* \vdash A$ and $A \vdash A * \top^*$		$A \vdash (A \multimap \mathbf{U}) \multimap \mathbf{U}$

where A, B, C, D are considered here to be propositional variables, and the named axioms are included in $\mathcal{A}_{\mathrm{BiBBI}}$ iff the BiBBI variant includes the corresponding property in Figure 1.

▶ **Lemma 5.7.** *Let $M = \langle W, \circ, \multimap, \triangledown, \mathrel{-\!\!\blacktriangleleft}, E, U\rangle$ be a modal frame satisfying axioms* (1)–(4) *of $\mathcal{A}_{BiBBI}$ in Definition 5.6. Then we have, for any $w, w_1, w_2 \in W$:*

$$w \in w_1 \multimap w_2 \;\Leftrightarrow\; w_2 \in w \circ w_1 \text{ and } w \in w_1 \mathrel{-\!\!\blacktriangleleft} w_2 \;\Leftrightarrow\; w_1 \in w_2 \triangledown w\ .$$

Given a modal frame $M = \langle W, \circ, \multimap, \triangledown, \mathrel{-\!\!\blacktriangleleft}, E, U\rangle$, we write $\ulcorner M \urcorner$ for $\langle W, \circ, E, \triangledown, U\rangle$.

▶ **Lemma 5.8.** *Let $M = \langle W, \circ, \multimap, \triangledown, \mathrel{-\!\!\blacktriangleleft}, E, U\rangle$ be a modal frame satisfying the set $\mathcal{A}_{BiBBI}$ of axioms corresponding to a BiBBI variant, as given by Definition 5.6. Then $\ulcorner M \urcorner$ is a BiBBI-model for that variant.*

Proof (sketch). First, $\ulcorner M \urcorner$ is a basic BiBBI-model, since it satisfies axioms (5)–(8) in Defn. 5.6. We just show that if an optional Sahlqvist axiom from Defn. 5.6 is valid in M, then M satisfies the corresponding frame property in Figure 1 (and thus $\ulcorner M \urcorner$ does too).

We just show the case of weak distribution here. Assume the weak distribution axiom of Definition 5.6 is valid in M and suppose that $(x_1 \circ x_2) \cap (y_1 \triangledown y_2) \neq \emptyset$. That is, we have $z \in (x_1 \circ x_2) \cap (y_1 \triangledown y_2)$ for some $z \in W$. We require to find a $w \in W$ such that $y_1 \in x_1 \circ w$ and $x_2 \in w \triangledown y_2$. Define a valuation ρ for M by $\rho(A) = \{x_1\}$, $\rho(B) = \{x_2\}$, $\rho(C) = \{y_1\}$ and $\rho(D) = \{y_2\}$. By construction, $z \models_\rho (A * B) \wedge (C \triangledown D)$. Since the weak distribution axiom is valid in M, we have that $z \models_\rho (A \wedge ((B \mathrel{-\!\!\circ} D) \multimap C)) * \top$. That is, for some z' we have $z' \models_\rho A \wedge ((B \mathrel{-\!\!\circ} D) \multimap C)$. Since $z' \models_\rho A$, we get $z' = x_1$ and so $x_1 \models_\rho (B \mathrel{-\!\!\circ} D) \multimap C$. As M satisfies axioms (1)–(4) by assumption, we can apply Lemma 5.7 to obtain w, w' such that $w' \in x_1 \circ w$ and $w \models_\rho B \mathrel{-\!\!\circ} D$ and $w' \models_\rho C$. As $w' \models_\rho C$, we have $y_1 \in x_1 \circ w$. Using Lemma 5.7 and commutativity of $\triangledown$ (forced by the validity of axiom (6) in M), we obtain from $w \models_\rho B \mathrel{-\!\!\circ} D$ that there exist w', w'' with $w'' \in w \triangledown w'$ and $w'' \models_\rho B$ and $w' \models_\rho D$. This means exactly that $x_2 \in w \triangledown y_2$ as required. This completes the proof. ◀

▶ **Definition 5.9.** We define a translation $t(-)$ from BiBBI-formulas to modal logic formulas, and a symmetric translation $u(-)$ in the opposite direction, by

$$\begin{array}{rclcrcl}
t(\phi) & = & \phi & \qquad & u(\phi) & = & \phi \\
t(\bot^*) & = & \neg\mathbf{U} & & u(\mathbf{U}) & = & \neg\bot^* \\
t(\neg A) & = & \neg t(A) & & u(\neg A) & = & \neg u(A) \\
t(A \mathbin{?} B) & = & t(A) \mathbin{?} t(B) & & u(A \mathbin{?} B) & = & u(A) \mathbin{?} u(B) \\
t(A \mathrel{-\!\!*} B) & = & \neg(t(A) \multimap \neg t(B)) & & u(A \multimap B) & = & \neg(u(A) \mathrel{-\!\!*} \neg u(B)) \\
t(A \mathbin{\text{⅋}} B) & = & \neg(\neg t(A) \triangledown \neg t(B)) & & u(A \triangledown B) & = & \neg(\neg u(A) \mathbin{\text{⅋}} \neg u(B)) \\
t(A \mathbin{\backslash^*} B) & = & t(A) \mathrel{-\!\!\circ} \neg t(B) & & u(A \mathrel{-\!\!\circ} B) & = & u(A) \mathbin{\backslash^*} \neg u(B)
\end{array}$$

where $\phi \in \{P, \top, \bot, \top^*\}$ and $? \in \{\wedge, \vee, \to, *\}$.

▶ **Lemma 5.10.** *If A is valid in some variant of BiBBI, then $t(A)$ is valid in the class of modal frames satisfying the corresponding Sahlqvist axioms $\mathcal{A}_{BiBBI}$ given by Definition 5.6.*

Proof (sketch). Let $M = \langle W, \circ, \multimap, \triangledown, \mathrel{-\!\!\circ}, E, U \rangle$ be a modal frame satisfying the axioms $\mathcal{A}_{\text{BiBBI}}$. By Lemma 5.8, $\ulcorner M \urcorner$ is a BiBBI-model for the variant of BiBBI determined by $\mathcal{A}_{\text{BiBBI}}$, and thus A is valid in $\ulcorner M \urcorner$. We require to show that $t(A)$ is valid in M, which follows by establishing the bi-implication $w \models_\rho A$ (in $\ulcorner M \urcorner$) $\Leftrightarrow$ $w \models_\rho t(A)$ (in M), for all $w \in W$ and valuations ρ. This bi-implication follows by structural induction on A, making use of Lemma 5.7 for the cases $A = B \mathrel{-\!\!*} C$ and $A = B \mathbin{\backslash^*} C$. ◀

▶ **Lemma 5.11.** *If B is provable in $\mathbf{ML}_{BiBBI} + \mathcal{A}_{BiBBI}$, then $u(B)$ is provable in the corresponding variant of BiBBI.*

Proof (sketch). We have to show that all the axioms and rules of normal modal logic (Defn. 5.3) and all the $\mathcal{A}_{\text{BiBBI}}$ axioms (Defn. 5.6) are derivable in the appropriate variant of BiBBI under the translation $u(-)$. For example, in the case of the Sahlqvist axiom for weak distribution from Defn. 5.6, we need to derive the following in BiBBI with weak distribution:

$$(u(A) * u(B)) \wedge \neg(\neg u(C) \mathbin{\text{⅋}} \neg u(D)) \vdash (u(A) \wedge \neg((u(B) \mathbin{\backslash^*} \neg u(D)) \mathrel{-\!\!*} \neg u(C)) * \top$$

The required derivations are often tedious and sometimes tricky: see [5] for details. ◀

▶ **Lemma 5.12.** *If $u(t(A))$ is provable in some variant of BiBBI then so is A.*

Proof (sketch). By structural induction on A. ◀

We may now finally prove our completeness result:

Proof of Theorem 3.8. Suppose A is valid in some BiBBI variant. By Lemma 5.10, $t(A)$ is then valid in the class of modal frames satisfying the Sahlqvist formulas $\mathcal{A}_{\text{BiBBI}}$ given by Defn. 5.6. By Theorem 5.5, $t(A)$ is provable in $\mathbf{ML}_{\text{BiBBI}} + \mathcal{A}_{\text{BiBBI}}$. Thus, by Lemma 5.11, $u(t(A))$ is provable in the corresponding variant of BiBBI. By Lemma 5.12, A is then provable in this BiBBI variant as required. ◀

6 Proof theory

In this section, we construct a cut-eliminating *display calculus* (cf. [1, 4, 3]) for BiBBI by combining a display calculus for classical logic with the display calculus for the multiplicative fragment of FILL given by Clouston et al [9]. Particular variants of BiBBI are handled via the inclusion or otherwise of optional structural rules.

▶ **Definition 6.1.** *Structures* are given by the following grammar, where F ranges over BiBBI-formulas: $X ::= F \mid \varnothing \mid \sharp X \mid X; X \mid X, X \mid X : X$. If X and Y are structures then $X \vdash Y$ is a *consecution*.

▶ **Definition 6.2.** For any structure Z we define the BiBBI-formulas Ψ_Z and Υ_Z by mutual structural induction:

$$\begin{array}{rclcrcl}
\Psi_F & = & F & \qquad & \Upsilon_F & = & F \\
\Psi_\varnothing & = & \top^* & & \Upsilon_\varnothing & = & \bot^* \\
\Psi_{\sharp X} & = & \neg\Upsilon_X & & \Upsilon_{\sharp X} & = & \neg\Psi_X \\
\Psi_{X;Y} & = & \Psi_X \wedge \Psi_Y & & \Upsilon_{X;Y} & = & \Upsilon_X \vee \Upsilon_Y \\
\Psi_{X,Y} & = & \Psi_X * \Psi_Y & & \Upsilon_{X,Y} & = & \Psi_X \mathrel{-\!\!*} \Upsilon_Y \\
\Psi_{X:Y} & = & \Psi_X \mathrel{\backslash\!\!^*} \Upsilon_Y & & \Upsilon_{X:Y} & = & \Upsilon_X \mathrel{\text{⅋}} \Upsilon_Y
\end{array}$$

Validity of the consecution $X \vdash Y$ (in a BiBBI variant) is then interpreted as validity of the formula $\Psi_X \vdash \Upsilon_Y$.

We give our display calculus DL_{BiBBI} for BiBBI in Figure 2. As usual, we give a set of *display postulates* written as a binary relation $<>_D$ on consecutions, and let *display-equivalence*, $\equiv_D$, be the reflexive-transitive closure of $<>_D$. Then, for any substructure occurrence Z in a consecution C, we can "display" Z as the entire antecedent or consequent as appropriate: that is, either $C \equiv_D Z \vdash X$ or $C \equiv_D X \vdash Z$ for some structure X (depending on whether Z occurs positively or negatively in C). For further details see e.g. [3].

The "variant" structural rules are included in DL_{BiBBI} only when we wish to consider particular variants of BiBBI. From left to right in Figure 2, the variant structural rules correspond respectively to: associativity; unit weakening; unit contraction; contraction; and weak distribution.

▶ **Lemma 6.3.** *For any structure X, both $X \vdash \Psi_X$ and $\Upsilon_X \vdash X$ are provable in* DL_{BiBBI}.

Proof (sketch). Structural induction on X. ◀

▶ **Theorem 6.4.** *$X \vdash Y$ is provable in a variant of* DL_{BiBBI} *if and only it is valid in the corresponding variant of BiBBI.*

Proof (sketch). For soundness, one just verifies directly that each rule of Figure 2 preserves validity, a straightforward exercise. For completeness, assume that $X \vdash Y$ is valid, i.e. that $\Psi_X \vdash \Upsilon_Y$ is a valid formula. By Theorem 3.8, $\Psi_X \vdash \Upsilon_Y$ is provable in the Hilbert system for (the required variant of) BiBBI. It is easy to show that the corresponding variant of DL_{BiBBI} can derive all principles of the Hilbert system, and thus $\Psi_X \vdash \Upsilon_Y$ is provable in DL_{BiBBI}. Then, using (Cut) and Lemma 6.3, we can prove $X \vdash Y$ in DL_{BiBBI} as required. ◀

Display postulates:

$$X;Y \vdash Z \;<\!>_D\; X \vdash \sharp Y;Z \;<\!>_D\; Y;X \vdash Z$$
$$X \vdash Y;Z \;<\!>_D\; X;\sharp Y \vdash Z \;<\!>_D\; X \vdash Z;Y$$
$$X \vdash Y \;<\!>_D\; \sharp Y \vdash \sharp X \;<\!>_D\; \sharp\sharp X \vdash Y$$

$$X,Y \vdash Z \;<\!>_D\; X \vdash Y,Z \;<\!>_D\; Y,X \vdash Z$$
$$X \vdash Y : Z \;<\!>_D\; X : Y \vdash Z \;<\!>_D\; X \vdash Z : Y$$

Identity rules:

$$\frac{}{P \vdash P}\,(\mathrm{Id}) \qquad \frac{X \vdash F \quad F \vdash Y}{X \vdash Y}\,(\mathrm{Cut}) \qquad \frac{X' \vdash Y'}{X \vdash Y}\;\; X \vdash Y \equiv_D X' \vdash Y'\;\; (\equiv_D)$$

Logical rules:

$$\frac{}{\bot \vdash X}\,(\bot\mathrm{L}) \quad \frac{\sharp F \vdash X}{\neg F \vdash X}\,(\neg\mathrm{L}) \quad \frac{F;G \vdash X}{F \wedge G \vdash X}\,(\wedge\mathrm{L}) \quad \frac{F \vdash X \quad G \vdash X}{F \vee G \vdash X}\,(\vee\mathrm{L}) \quad \frac{X \vdash F \quad G \vdash Y}{F \to G \vdash \sharp X;Y}\,(\to\mathrm{L})$$

$$\frac{}{X \vdash \top}\,(\top\mathrm{R}) \quad \frac{X \vdash \sharp F}{X \vdash \neg F}\,(\neg\mathrm{R}) \quad \frac{X \vdash F \quad X \vdash G}{X \vdash F \wedge G}\,(\wedge\mathrm{R}) \quad \frac{X \vdash F;G}{X \vdash F \vee G}\,(\vee\mathrm{R}) \quad \frac{X;F \vdash G}{X \vdash F \to G}\,(\to\mathrm{R})$$

$$\frac{\varnothing \vdash X}{\top^* \vdash X}\,(\top^*\mathrm{L}) \quad \frac{F,G \vdash X}{F * G \vdash X}\,(*\mathrm{L}) \quad \frac{X \vdash F \quad G \vdash Y}{F \mathrel{-\!\!*} G \vdash X,Y}\,(\mathrel{-\!\!*}\mathrm{L}) \quad \frac{}{\bot^* \vdash \varnothing}\,(\bot^*\mathrm{L}) \quad \frac{F \vdash X \quad G \vdash Y}{F \mathbin{\diamondsuit} G \vdash X : Y}\,(\diamondsuit\mathrm{L})$$

$$\frac{}{\varnothing \vdash \top^*}\,(\top^*\mathrm{R}) \quad \frac{X \vdash F \quad Y \vdash G}{X,Y \vdash F * G}\,(*\mathrm{R}) \quad \frac{X,F \vdash G}{X \vdash F \mathrel{-\!\!*} G}\,(\mathrel{-\!\!*}\mathrm{R}) \quad \frac{X \vdash \varnothing}{X \vdash \bot^*}\,(\bot^*\mathrm{R}) \quad \frac{X \vdash F : G}{X \vdash F \mathbin{\diamondsuit} G}\,(\diamondsuit\mathrm{R})$$

$$\frac{F : G \vdash X}{F \mathbin{\backslash^*} G \vdash X}\,(\backslash^*\mathrm{L}) \qquad \frac{X \vdash F \quad G \vdash Y}{X : Y \vdash F \mathbin{\backslash^*} G}\,(\backslash^*\mathrm{R})$$

Structural rules:

$$\frac{X \vdash Z}{X;Y \vdash Z}\,(\mathrm{Wk}) \quad \frac{X;X \vdash Z}{X \vdash Z}\,(\mathrm{Ctr}) \quad \frac{W,(X,Y) \vdash Z}{(W,X),Y \vdash Z}\,(*\mathrm{A}) \quad \frac{X \vdash Y}{\varnothing,X \vdash Y}\,(\varnothing\mathrm{WkL}) \quad \frac{\varnothing,X \vdash Y}{X \vdash Y}\,(\varnothing\mathrm{CtrL})$$

Variant structural rules:

$$\frac{W \vdash (X : Y) : Z}{W \vdash X : (Y : Z)}\,(\diamondsuit\mathrm{A}) \quad \frac{X \vdash Y}{X \vdash Y : \varnothing}\,(\varnothing\mathrm{WkR}) \quad \frac{X \vdash Y : \varnothing}{X \vdash Y}\,(\varnothing\mathrm{CtrR}) \quad \frac{X \vdash Y : Y}{X \vdash Y}\,(\diamondsuit\mathrm{Ctr})$$

$$\frac{W,(X : Y) \vdash Z}{(W,X) : Y \vdash Z}\,(\mathrm{WDist})$$

Figure 2 The proof rules of $\mathrm{DL}_{\mathrm{BiBBI}}$. W, X, Y, Z range over structures, F, G range over BiBBI-formulas and P ranges over $\mathcal{V}$.

▶ **Theorem 6.5.** *Any* DL_{BiBBI} *proof of* $X \vdash Y$ *can be transformed into a proof of* $X \vdash Y$ *without (Cut).*

Proof (sketch). We just verify that the proof rules of $\mathrm{DL}_{\mathrm{BiBBI}}$ collectively satisfy Belnap's well known cut-elimination conditions (C2)–(C8) [1]. The verification is straightforward, and similar to the one carried out in [3]. ◀

7 Conclusions

In this paper, we study "sub-classical" bunched logics between BBI and CBI, where a multiplicative "disjunction family" of connectives, $(\mathbin{\dot{\vee}}, \mathbin{\dot{-}\!\!*}, \dot{\bot}^*)$, exists alongside the usual "conjunction family" $(*, -\!\!*, \top^*)$. The two families are dual to one another in an intuitionistic sense: $*$ and $\mathbin{\dot{\vee}}$ are related, if at all, not by De Morgan equivalence but by the *weak distribution* law, $A * (B \mathbin{\dot{\vee}} C) \vdash (A * B) \mathbin{\dot{\vee}} C$. From the point of view of linear logic, the variants of our BiBBI can be seen as free combinations of classical logic with various multiplicative fragments of Hyland and de Paiva's FILL [16].

We have given a Kripke frame semantics for our logic(s) in which various logical axioms of FILL have natural semantic correspondents as first-order conditions on BiBBI-models (cf. Figure 1). We provide a completeness proof for this semantics, based on the Sahlqvist completeness theorem for modal logic, and moreover we obtain completeness for *any* variant of BiBBI given by a choice of logical principles from Figure 1.

Investigating the models of our sub-classical bunched logics in more detail, we find that heap-like models of BiBBI, as used in separation logic, can be obtained by interpreting $\mathbin{\dot{\vee}}$ using natural notions of heap *intersection*. (This stands in contrast to the situation for classical bunched logic CBI, of which heaps are not models.) In such models, the above weak distribution law holds, but this unavoidably comes at the expense of the unit law $A \mathbin{\dot{\vee}} \dot{\bot}^* \equiv A$ (see Prop. 4.3). However, this is not true of all interesting models of BiBBI; we show how to turn sufficiently well-behaved BBI-models (such as the heap model) into more complex BiBBI-models in which both weak distribution and the unit law hold, based on pairing every world in the original model with a larger "environment" (Theorem 4.13).

We are cautiously optimistic that the disjunctive machinery of BiBBI might usefully be applied to program verification based on separation logic. As in linear logic, it seems more difficult to reason intuitively using multiplicative disjunction than using multiplicative conjunction. However, the fact that disjunction can be interpreted using natural heap intersection operations, which are closely related to the union operation used to reason about algorithms with complex sharing [15, 12], leads us to hope that such intuitions are within reach. We hope to explore this direction further in future work.

References

1 Nuel D. Belnap, Jr. Display logic. *Journal of Philosophical Logic*, 11:375–417, 1982.
2 Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic*. Cambridge University Press, 2001.
3 James Brotherston. Bunched logics displayed. *Studia Logica*, 100(6):1223–1254, 2012.
4 James Brotherston and Cristiano Calcagno. Classical BI: Its semantics and proof theory. *Logical Methods in Computer Science*, 6(3), 2010.
5 James Brotherston and Jules Villard. Bi-intuitionistic boolean bunched logic. Technical Report RN/14/06, University College London, 2014.
6 James Brotherston and Jules Villard. Parametric completeness for separation theories. In *Proc. POPL-41*, pages 453–464. ACM, 2014.

7 Cristiano Calcagno, Dino Distefano, Peter O'Hearn, and Hongseok Yang. Compositional shape analysis by means of bi-abduction. *Journal of the ACM*, 58(6), December 2011.

8 Cristiano Calcagno, Philippa Gardner, and Uri Zarfaty. Context logic as modal logic: Completeness and parametric inexpressivity. In *Proc. POPL-34*, pages 123–134. ACM, 2007.

9 Ranald Clouston, Jeremy Dawson, Rajeev Goré, and Alwen Tiu. Annotation-free sequent calculi for full intuitionistic linear logic. In *Proc. CSL-22*, pages 197–214. Dagstuhl, 2013.

10 Robert Dockins, Aquinas Hobor, and Andrew W. Appel. A fresh look at separation algebras and share accounting. In *Proc. APLAS-7*, pages 161–177. Springer, 2009.

11 Didier Galmiche and Dominique Larchey-Wendling. Expressivity properties of Boolean BI through relational models. In *Proc. FSTTCS-26*, pages 357–368. Springer, 2006.

12 Philippa Gardner, Sergio Maffeis, and Gareth David Smith. Towards a program logic for JavaScript. In *Proc. POPL-39*, pages 31–44, 2012.

13 Jean-Yves Girard and Yves Lafont. Linear logic and lazy computation. In *Proc. TAPSOFT*, pages 52–66. Springer-Verlag, 1987.

14 Alexey Gotsman, Byron Cook, Matthew Parkinson, and Viktor Vafeiadis. Proving that non-blocking algorithms don't block. In *Proc. POPL-36*, pages 16–28. ACM, 2009.

15 Aquinas Hobor and Jules Villard. The ramifications of sharing in data structures. In *Proc. POPL-40*, pages 523–536. ACM, 2013.

16 Martin Hyland and Valeria de Paiva. Full intuitionistic linear logic (extended abstract). *Annals of Pure and Applied Logic*, 64(3):273—291, 1993.

17 Samin Ishtiaq and Peter W. O'Hearn. BI as an assertion language for mutable data structures. In *Proc. POPL-28*, pages 14–26. ACM, 2001.

18 Peter W. O'Hearn and David J. Pym. The logic of bunched implications. *Bulletin of Symbolic Logic*, 5(2):215–244, 1999.

19 David Pym. *The Semantics and Proof Theory of the Logic of Bunched Implications*. Applied Logic Series. Kluwer, 2002.

20 David Pym, Peter O'Hearn, and Hongseok Yang. Possible worlds and resources: The semantics of BI. *Theor. Comp. Sci.*, 315(1):257–305, 2004.

21 John C. Reynolds. Separation logic: A logic for shared mutable data structures. In *Proc. LICS-17*, pages 55–74. IEEE, 2002.

22 H. Yang, O. Lee, J. Berdine, C. Calcagno, B. Cook, D. Distefano, and P. O'Hearn. Scalable shape analysis for systems code. In *Proc. CAV-20*, pages 385–398. Springer, 2008.

Classical and Intuitionistic Arithmetic with Higher Order Comprehension Coincide on Inductive Well-Foundedness

Stefano Berardi

Università di Torino, Italy

Abstract

Assume that we may prove in Arithmetic with Comprehension axiom that a primitive recursive binary relation R is well-founded, using the inductive definition of well-founded. In this paper we prove that the proof that R is well-founded may be made intuitionistic. Our result generalizes to any implication between such formulas. We conclude that if we are able to formulate any mathematical problem as the fact that some primitive recursive relation is well-founded, then intuitionistic and classical provability coincide, and for such a statement we may always find an intuitionistic proof, if we may find a proof at all.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases Intuitionism, Inductive Definitions, Proof Theory, impredicativity, omega rule

Digital Object Identifier 10.4230/LIPIcs.CSL.2015.343

1 Introduction. A conservativity result for the statements of inductive well-foundedness

The proof principle of induction, originally called *transfinite induction*, is credited to the founder of intuitionism, Brouwer ([19]). In 1967 Howard and Kreisel [10] remarked that (transfinite) induction is the most useful formulation of well-foundedness in an intuitionistic context. In 1971, P. Martin-Löf studied an intuitionistic natural deduction version of transfinite induction [13]. By building over their work, many classical theorems, whose original version is not intuitionistically provable, have been reformulated in such a way to become intuitionistically provable. For some primitive recursive relations $R_1, \ldots, R_n, R$ over `Nat`, the new statements have the following form:

(1) "if $R_1, \ldots, R_n$ are inductively well-founded, then R is inductively well-founded"

Among many examples we recall: Higman Lemma [5], [7], compactness results in formal topology [6], Ramsey Theorem in Combinatorial Mathematics, the Termination Theorem and the Size-Change Termination Theorem in Computer Science [3], [20].

The results of this paper are an *a posteriori* justification of the existence of these intuitionistic results. We guarantee, in the case of a classical proof of a statement of the form (1), that constructivization is always possible in principle, at least for the arithmetical proofs using Comprehension of all finite orders: second order arithmetic, third order and so forth. Comprehension axiom for sets of natural numbers says that any predicate on natural numbers defines a set, Comprehension axiom for sets of sets of natural numbers says that any predicate on sets of natural numbers defines a set of sets, and so forth, for all finite orders. Let us call *impredicative* a proof possibly using Comprehension axiom of some finite

24th EACSL Annual Conference on Computer Science Logic (CSL 2015).
Editor: Stephan Kreutzer; pp. 343–358

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

order. We prove that any impredicative proof of a statement of the form (1) may avoid the use of Excluded Middle. Often the resulting proof will be intuitionistic but impredicative.

We claim having an effective method for transforming classical proofs into intuitionistic ones, however, we do not claim that this method is feasible in practice. We do not claim we found an optimal proof, either. Our meta-proof that a result is intuitionistically provable uses intuitionistic but impredicative meta-reasoning. We are not yet able to define some simple effective procedure to turn classical proofs of well-foundedness into intuitionistic ones. We tried with Gödel's $\neg\neg$-translation and with Friedman's A-translation [8], but they did not work.

There are results along the same direction. Sieg proved in his ph.d. thesis that whenever ω-iterated inductive definitions prove that a primitive recursive ordinal is well-founded, then the ordinal belongs to some primitive recursive denotation system, which may be proved to be well-founded intuitionistically. An account of his proof may be found in [18].

Here we address a more general case: a classical proof using the Comprehension axioms of all finite orders. The conclusion of the classical proof has the more general form: if R_1, $\ldots$, R_n are inductively well-founded, then R is inductively well-founded, with $R_1, \ldots, R_n, R$ primitive recursive relations over $\mathtt{Nat}$.

Our result may be seen as a generalization of the well-known result of conservativity of classical analysis w.r.t. intuitionistic analysis for Π^0_2-formulas by Kreisel [12]. This conservativity results generalizes to any implications between Π^0_2-formulas (this is an old remark, for a proof see for instance [4]). We extend the conservativity result to all implications among Π^1_1-formulas, provided these formulas may be expressed through the well-foundedness of some primitive recursive relation, and using the inductive definition of well-foundedness. The predicate "being the code of a primitive recursive well-founded relation" is Π^1_1-complete (see for instance [9]). Thus, our result allows to reformulate any classical theorem which is an implication between Π^1_1-formulas by some classically equivalent and intuitionistically provable implication between statements of inductive well-foundedness. The original classical theorem may be not intuitionistically provable.

This is the plan of the paper. In Section 2 we introduce $\mathtt{PA}^\omega$, classical arithmetic with Comprehension of order ω, the formal system we deal with in this paper. In Section 3 we describe the inductive definition of well-founded relation and its basic properties in Intuitionistic Arithmetic. In Section 4 we sketch the proof idea for the conservativity result, and in §5 we prove it. In Section 6 we draw our conclusions and we provide some examples. The acknowledgments are in Paragraph 6.1.

2 The conservativity result and the formal system for Classical Higher Order Arithmetic

Assume $R_1, \ldots, R_n, R$ are some primitive recursive binary relations on $\mathtt{Nat}$. Let us denote with $\mathtt{WF}(R)$ the statement "R is inductively well-founded" (to be defined later). We want to prove: whenever we have a proof of $\mathtt{WF}(R_1), \ldots, \mathtt{WF}(R_n) \implies \mathtt{WF}(R)$ in Classical Peano Arithmetic with Comprehension axiom of order ω, then we may prove $\mathtt{WF}(R_1), \ldots, \mathtt{WF}(R_n) \implies \mathtt{WF}(R)$ in the "Theory of Species" [14]. The "Theory of Species" is a natural deduction with higher order predicate variables, and Comprehension axiom of level ω hidden in the $\forall$-elimination rule. Comprehension of level two (for sets of natural numbers) proves Paris-Harrington Theorem, the Theorem about Goodstein Sequences, Higman Lemma, Kruskal Lemma, which are not provable in first order arithmetic (classical or intuitionistic). Each level of comprehension adds new theorems. In spite of these strong assumptions, the "Theory of Species" is an

intuitionistic logic, because it has the disjunctive property and the witness property ([14], p. 231, Section 11.1, 11.2). In fact, it may be considered as *the Intuitionistic Arithmetic with Comprehension axiom of order* ω.

In this paper we are interested in the mere existence of an intuitionistic formal proof, and not in an efficient way of writing of it. We prove our result using intuitionistic but impredicative meta-reasoning. For the moment, we do not have a meta-proof using first order arithmetical reasoning.

We first define a sequent calculus version of the Intuitionistic Theory of Species plus Excluded Middle. We call this system *the Classical Arithmetic with Comprehension axiom of order* ω. Following [13], [16], we use a superscript to denote the level of comprehension, and we call this system $\mathtt{PA}^\omega$. $\mathtt{PA}^\omega$ should not be confused with $\mathtt{PA}_\omega$, the system for first order classical arithmetic having higher order functions and no Comprehension axiom. We do not have higher order functions instead. We do have higher order predicate types: $\mathtt{Nat} \to \mathtt{Prop}$ (the type of sets of natural numbers), $(\mathtt{Nat} \to \mathtt{Prop}) \to \mathtt{Prop}$ (the type of sets of sets of natural numbers), and so forth. Comprehension for second order, third order, ..., arithmetic is the statement $\exists X^{\sigma\to\mathtt{Prop}}.(\forall x^\sigma.X(x) \Leftrightarrow P)$, for $\sigma = \mathtt{Nat}, (\mathtt{Nat} \to \mathtt{Prop}), \ldots$. Comprehension is not an axiom of our sequent calculus $\mathtt{PA}^\omega$, but we will express it through left-introduction rules for universal quantifiers: $\forall X^{\mathtt{Nat}\to\mathtt{Prop}}.A$, $\forall X^{(\mathtt{Nat}\to\mathtt{Prop})\to\mathtt{Prop}}.A$, ... This is analogous to the way Martin-Lof expresses Comprehension axiom through the $\forall$-elimination rule. We choose this formalization of Comprehension in order to use Girard's candidate method for deriving normalization of $\mathtt{PA}^\omega$.

▶ **Definition 1** (The Language of $\mathtt{PA}^\omega$). $\mathtt{PA}^\omega$ is the formal system having:

1. One numeral for each $n \in \mathtt{Nat}$, one symbol for each primitive recursive function, infinitely many variables $x^{\mathtt{Nat}}, y^{\mathtt{Nat}}, z^{\mathtt{Nat}}, \ldots$ of type $\mathtt{Nat}$. All terms we may define from them.
2. *Predicates* types: the type $\mathtt{Prop}$ of formulas, and with σ, τ also $\sigma \to \tau$ and $\mathtt{Nat} \to \tau$. If σ is a predicate type, its degree $\deg(\sigma)$ is inductively defined by $\deg(\mathtt{Nat}) = 1$, $\deg(\mathtt{Prop}) = 2$ and $\deg(\sigma \to \tau) = \max(\deg(\sigma) + 1, \deg(\tau))$.
3. One predicate constant p for each primitive recursive predicate on $\mathtt{Nat}$ of any arity, infinitely many variables $X^\sigma, Y^\sigma, Z^\sigma, \ldots$ for each predicate type σ.
4. All formulas we may define from the atomic formulas $p(t_1, \ldots, t_n), X(t_1, \ldots, t_n)$ using the connective $\to$, and two kinds of $\forall$: quantification over $\mathtt{Nat}$, and for any predicate type σ, quantification over σ.
5. All predicates we may define from formulas by simply typed λ-abstraction.
6. As predicate equality, the smallest equivalence relation including: the β-reduction and the reduction replacing a closed term $t : \mathtt{Nat}$ by the numeral it denotes.
7. Two-sided sequents $\Gamma \vdash \Delta$, for any finite sets of formulas Γ, Δ.

The degree of a type is inductively defined by $\deg(\mathtt{Nat}) = 1$, $\deg(\mathtt{Prop}) = 2$, $\deg(\sigma \to \tau) = \max(1 + \deg(\sigma), \deg(\tau))$. The order of a formula is the maximum degree of the types of its variables: first order formulas have integer variables, second order formulas have integer variables and variables on sets of integers, and so forth. For any predicate type σ, the formula $\exists X^\sigma.A$ is not a primitive formula of the language but it is expressed through its canonical higher order definition: $\forall Z^{\mathtt{Prop}}.(\forall X^\sigma.(A \to Z^{\mathtt{Prop}})) \to Z^{\mathtt{Prop}}$, for $Z^{\mathtt{Prop}}$ not free in A. In the same way we define $A \vee B$ as $\forall Z^{\mathtt{Prop}}.(A \to Z^{\mathtt{Prop}}), (B \to Z^{\mathtt{Prop}}) \to Z^{\mathtt{Prop}}$, and $A \wedge B$ as $\forall Z^{\mathtt{Prop}}.(A, B \to Z^{\mathtt{Prop}}) \to Z^{\mathtt{Prop}}$, for $Z^{\mathtt{Prop}}$ not free in A, B, and $A \Leftrightarrow B$ as $(A \to B) \wedge (B \to A)$. Thus, we may now write Comprehension $\exists X^{\sigma\to\mathtt{Prop}}.(\forall x^\sigma.X(x) \Leftrightarrow P)$ as a defined formula. If $\deg(\sigma) = n$ we say that this formula is Comprehension of order n.

We will identify a numeral with the number it denotes. We often skip the type superscript of a variable, but we always use lower case letters $x, y, z, \ldots$ for natural number variables

and upper case letters $X, Y, Z, \ldots$ for predicate variables. We consider any recursively enumerable set of basic arithmetical axioms. Any basic arithmetical axiom should have the form $\alpha_1, \ldots, \alpha_n \vdash \alpha$ or $\beta_1, \ldots, \beta_n \vdash$, with each formula of the form $p(t_1, \ldots, t_n)$ for some primitive recursive p and some terms $t_1, \ldots, t_n$. Basic arithmetical axioms should include $t + 1 = u + 1 \vdash t = u$ and $0 = t + 1 \vdash \emptyset$, all rules making $=$ an equivalence relation, and all definition rules for all primitive recursive maps and relations. Basic arithmetical axioms should be true, and should be closed under substitution and cut. The last clause is required if we want to have full cut-elimination for $\mathtt{PA}^\omega$, otherwise cuts between axioms cannot be eliminated.

Proof rules are those of two-sided sequent calculus, with left-introduction rules for $\forall X^\sigma.A$ expressing comprehension of any finite order.

▶ **Definition 2** (Proofs of $\mathtt{PA}^\omega$)**.** Proof trees of $\mathtt{PA}^\omega$ are exactly the *finite trees* built by the rules below, where Γ, Δ are finite sets of formulas, A, B are formulas, $\alpha_1, \ldots, \alpha_n \vdash \alpha$ and $\beta_1, \ldots, \beta_n \vdash$ are basic arithmetical axioms.

$$\frac{}{\Gamma, \alpha_1, \ldots, \alpha_n \vdash \Delta, \alpha}\ (\mathtt{axiom})$$

$$\frac{}{\Gamma, \beta_1, \ldots, \beta_n \vdash \Delta}\ (\mathtt{axiom})$$

$$\frac{}{\Gamma, A \vdash \Delta, A}\ (\mathtt{id})$$

$$\frac{\Gamma_1 \vdash \Delta_1, A \quad \Gamma_2, A \vdash \Delta_2}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}\ (\mathtt{cut})$$

$$\frac{\Gamma, A \implies B \vdash \Delta, A \quad \Gamma, A \implies B, B \vdash \Delta}{\Gamma, A \implies B \vdash \Delta}\ (\implies_l)$$

$$\frac{\Gamma, A \vdash \Delta, A \implies B, B}{\Gamma \vdash \Delta, A \implies B}\ (\implies_r)$$

$$\frac{\Gamma, \forall xA, A[t/x] \vdash \Delta}{\Gamma, \forall xA \vdash \Delta}\ (\text{First Order } \forall_l)$$

$$\frac{\Gamma \vdash \Delta, \forall xA, A[z/x] \quad (z \notin \mathtt{FV}(\Gamma \vdash \Delta, \forall xA))}{\Gamma \vdash \Delta, \forall xA}\ (\text{First Order } \forall_r)$$

$$\frac{\Gamma, \forall X^\sigma A, A[P/X^\sigma] \vdash \Delta}{\Gamma, \forall X^\sigma A \vdash \Delta}\ (\text{Higher Order } \forall_l)$$

$$\frac{\Gamma \vdash \Delta, \forall XA, A[Z/X] \quad (Z \notin \mathtt{FV}(\Gamma \vdash \Delta, \forall XA))}{\Gamma \vdash \Delta, \forall XA}\ (\text{Higher Order } \forall_r)$$

$$\frac{\Gamma \vdash \Delta, \forall xA, A[0/x] \quad \Gamma, A[z/x] \vdash \Delta, \forall xA, A[z+1/x] \quad (z \notin \mathtt{FV}(\Gamma \vdash \Delta, \forall xA))}{\Gamma \vdash \Delta, \forall xA}\ (\mathtt{ind})$$

We say that a $\forall_l$-rule for a formula $\forall X^\sigma A$ has order n if $\deg(\sigma) = n$. Choose any predicate type σ with $\deg(\sigma) = n$. Then the sequent $\vdash \exists X^{\sigma \to \mathtt{Prop}}.(\forall x^\sigma.X(x) \Leftrightarrow P)$ expresses Comprehension axiom of level n. We claim that there is a proof of this sequent using $\forall_l$-rules of order $\leq n + 1$ only. For instance, the rules $\forall_l$ of order ≤ 2 derive Comprehension of order 1, or Comprehension for sets of integers. We call order n Peano Arithmetic, and we denote by $\mathtt{PA}^n$, the system with $\forall_l$ restricted to the order $\leq n$. For instance, First Order Arithmetic $\mathtt{PA}^1$ derives no Comprehension axiom, while Second Order Arithmetic $\mathtt{PA}^2$ only derives Comprehension of order 1 (for sets of natural numbers).

In the identity rule and in the cut rule, formulas are identified up to predicate equality (Def. 1.5). In all rules but `cut` the conclusion of the rule is included in each premise (if any).

The relation R we fixed, being a primitive recursive binary predicate, is a symbol of the language of $\mathtt{PA}^\omega$. Let us assume $\mathtt{PA}^n \vdash \mathtt{WF}(R)$. We will intuitionistically prove $\mathtt{WF}(R)$ using Comprehension of order $n + 1$.

This is the proof idea. We first define an infinitary semi-formal proof system, then we interpret the finitary proof system in the infinitary one and we use the infinitary system to derive our result.

▶ **Definition 3** (ω-rule)**.** Let $\mathtt{PA}^\omega$ + recursive ω-rule be the semi-formal system obtained from $\mathtt{PA}^\omega$ by:

1. considering only the sequents with *no free number variables*,
2. replacing the rules $\mathtt{ind}$ and $\forall_r$ for $\mathtt{Nat}$ with the recursive ω-rule: derive $\Gamma \vdash \Delta, \forall x^{\mathtt{Nat}}.A$ from a recursively given family of proof-trees, one proof of $\Gamma \vdash \Delta, \forall x A, A[m/x^{\mathtt{Nat}}]$ for each numeral $m \in \mathtt{Nat}$.

We write $\pi : \Gamma \vdash \Delta$ for: π is a well-founded recursive proof-tree with ω-rule of conclusion $\Gamma \vdash \Delta$. Let $n > 0$. $\mathtt{PA}^n$ + recursive ω-rule is $\mathtt{PA}^\omega$ + recursive ω-rule with the $\forall_l$-rules restricted to all orders $\leq n$.

The ω-rule may be represented as follows:

$$\frac{\ldots \quad \Gamma \vdash \Delta, \forall x A, A[m/x] \quad \ldots \quad (\text{for all } m \in \mathtt{Nat})}{\Gamma \vdash \Delta, \forall x A} \ (\omega\text{-rule})$$

Proofs of $\mathtt{PA}^\omega$ + recursive ω-rule are at most countable well-founded recursive trees decorated with the rules of the system. A proof tree is called *normal* if it has no cut rule. It is well-known that there is an effective method for turning any proof of any A in $\mathtt{PA}^n$ into a proof of A in $\mathtt{PA}^n$ + recursive ω-rule. It is also well-known that the system $\mathtt{PA}^n$ + recursive ω-rule has a normalization algorithm, turning any proof-tree of $\Gamma \vdash \Delta$ into a normal proof-tree, and that the proof may be expressed using order $n + 1$ Comprehension.

3 The inductive definition of a well-founded relation

In Intuitionistic Arithmetic, we define well-founded relations through an inductive definition. Let R be any binary relation over a set I. Classically, the usual definition of "R is well-founded" says: "there is no infinite R-decreasing chain", or: "all R-decreasing chains are finite". In Intuitionistic Arithmetic, these statements are not informative enough, and we prefer the inductive definition of well-foundedness, which runs as follows. A predicate X over a set I is called R-inductive if X contains x whenever X contains all y such that yRx. We say that I, R (or just R for short) is inductively well-founded, and we write $\mathtt{WF}(R)$, if all R-inductive properties X are true for all $x \in I$. By definition unfolding, we obtain:

$$\mathtt{WF}(R) = \forall x \in I. \forall X. (\forall y \in I. (\forall z \in I. zRy \implies X(z)) \implies X(y)) \implies X(x)$$

From now on, we will often write "well-founded" as a shorthand for "inductive well-founded". If (I, R) is well-founded, then, in order to prove that $\forall x \in I. P(x)$, it is enough to prove that P is R-inductive. This well-known proof method is called "by induction on x and R". For instance, when R is the successor relation on $\mathtt{Nat}$, we re-obtain induction on $\mathtt{Nat}$. Induction may be nested. In order to prove $P(x)$ we may use induction again, over some set I_x and some relation R_x, possibly depending on $x \in I$. In this case we speak of "secondary induction on $y \in I_x$ and R_x".

Our conservativity proofs deals with the sub-formulas of $\mathtt{WF}(R)$. Now we will list them, just skipping the atomic sub-formulas and the implications of atomic formulas. If we list the sub-formulas of $\mathtt{WF}(R)$ from right to left, the first four sub-formulas we find are atomic, or implications of atomic. The fifth sub-formula, and the first in our list, is: *(1)* $\forall z \in I. zRy \implies X(z)$. Then we find, in this order: *(2)* $(\forall z \in I. zRy \implies X(z)) \implies X(y)$, then *(3)* $\forall y \in I. (\forall z \in I. zRy \implies X(z)) \implies X(y)$, then *(4)* $(\forall y \in I. (\forall z \in I. zRy \implies X(z)) \implies X(y)) \implies X(x)$, then *(5)* $\forall X. (\forall y \in I. (\forall z \in I. zRy \implies X(z)) \implies X(y)) \implies X(x)$, and eventually *(6)* the formula $\mathtt{WF}(R)$ itself. These 6 expressions are

cumbersome, therefore we will introduce a name for each of them. There is nothing to understand here: we assign 6 names (including $\mathtt{WF}(R)$ itself) we will use later.

▶ **Definition 4** (Formal definition of (inductive) well-foundedness)**.** Let R be any binary relation over a set I and $X : I \to \mathtt{Prop}$ any variable of unary predicate over I.

1. $\mathtt{IH}(y, X) = \forall z \in I.(zRy \implies X(z))$ *(R-inductive hypothesis in y for X)*
2. $\mathtt{Ind}(y, X) = (\mathtt{IH}(y, X) \implies X(y))$ *(X is R-inductive in y)*
3. $\mathtt{IND}(X) = \forall y \in I.\mathtt{Ind}(y, X)$ *(X is R-inductive)*
4. $\mathtt{wf}(x, X) = (\mathtt{IND}(X) \implies X(x))$ *(x is R-well-founded w.r.t. X)*
5. $\mathtt{Wf}(x) = \forall X.\mathtt{wf}(x, X)$ *(x is R-well-founded)*
6. $\mathtt{WF}(R) = \forall x \in I.\mathtt{Wf}(x)$ *((I, R) is well-founded)*

We abbreviate "(I, R) is well-founded" by "R is well-founded" when I is clear from the context. The predicates defined in the points $1 - 5$ above, to be accurate, should be written with an extra argument R. We skipped R because it is fixed and would clutter our formulas uselessly. We left the argument X, even if X is fixed, as a memo for the name of the variable X denoting a generic unary predicate on I in the formula $\mathtt{WF}(R)$.

ω-rule is complete w.r.t. the statements of the form $\mathtt{WF}(R)$.We may state this fact as follows. Recall that $\pi : \Gamma \vdash \Delta$ denotes that π is a well-founded recursive proof-tree of conclusion $\Gamma \vdash \Delta$.

▶ **Lemma 5** (ω-rule is complete for $\mathtt{WF}(R)$)**.** *There is some recursive family of trees indexed over binary primitive recursive relations over* $\mathtt{Nat}$*, of the form* $\{\pi_R | R$ *bin.prim.rec.*$\}$*, such that:*

$$\mathtt{WF}(R) \implies \pi_R : \mathtt{WF}(R)$$

This fact is provable in Second Order Intuitionistic Arithmetic.

Classically and using choice, the inductive definition of $\mathtt{WF}(R)$ is equivalent to the classical formulation which says "all descending R-chains are finite". This equivalence is not provable in Higher Order Intuitionistic Arithmetic.

A method for proving well-foundedness is the simulation of a relation into another well-founded relation. Informally, we may simulate $x \in I$ by $y \in J$ using a relation $\sim$ if, by moving from I to J using $\sim$, we may simulate R-chains through S-chains. Our method is an intuitionistic and simplified version of the method used for proving that a labeled state transition systems strongly terminates [15], if we take as set of labels of a transition a singleton.

▶ **Definition 6** (Simulation)**.** Assume R is a binary relation over a set I and S is a binary relation over a set J. Assume $\sim \subseteq I \times J$.

1. $\sim$ is a simulation of (I, R) into (J, S) if whenever $x \sim y$ and $x'Rx$ then for some $y' \in J$ we have $y'Sy$ and $x' \sim y'$.
2. $\sim$ is a bisimulation if both $\sim$ and $\sim^{-1}$ are simulations.
3. $\sim$ is a weak simulation of (I, R) into (J, S) if whenever $x \sim y$ and $x'Rx$ then for some $y' \in J$ we have: $y'Sy$ and either $x \sim y'$ or $x' \sim y'$.

In the rest of the paper we will use some well-known definitions and facts about well-founded relations and simulations.

We first state the Kleene-Brouwer Theorem. We denote by $\sigma * \tau$ the concatenation of two finite sequences σ, τ, and with $\sigma @ n = \sigma * \langle n \rangle$ the appending of an element to a list. We denote the one-step extension relation on finite sequences by $\prec_1$, and we define it by $\sigma @ n \prec_1 \sigma$. We denote by $\prec$ the strict prefix relation, and we define it by $\sigma * \langle n \rangle * \tau \prec \sigma$. We denote by

$\prec\!\!\prec$ the post-order, a total order over the finite sequences over `Nat`. $\prec\!\!\prec$ is defined as follows: $\sigma@n*\tau \prec\!\!\prec \sigma$, and if $n' < n$ then $\sigma@n'*\tau \prec\!\!\prec \sigma@n*\rho$. Then the Kleene-Brouwer Theorem may be stated as follows: "if $(T, \prec_1)$ is a well-founded tree of finite sequences over `Nat`, then $(T, \prec\!\!\prec)$ is well-founded".

The main property of simulations we need to prove is: if $y \in J$ simulates some $x \in I$ and y is S-well-founded, then x is R-well-founded. Recall that "well-founded" is short for "inductively well-founded" here.

All these results have an intuitionistic proof, included in the next Lemma.

▶ **Lemma 7.** *Properties of well-founded relations] Assume T is any tree of finite sequences over some set K, with x child of y if and only if $x \prec_1 y$. Assume I, J are any sets, and R, S are any binary relations, respectively, on I and on J.*

1. **(a)** *Well-foundedness is an inductive predicate: for any $x \in I$ we have* $\forall y.(yRx \implies y \text{ is } R\text{-well-founded}) \implies x \text{ is } R\text{-well-founded}$
   (b) *The converse holds: for any $x \in I$ we have* $x \text{ is } R\text{-well-founded} \implies \forall y.(yRx \implies y \text{ is } R\text{-well-founded})$
2. **(a)** *A tree is well-founded if and only if all proper descendants of the root of the tree are well-founded.*
   (b) *A tree is well-founded if and only if its root is.*
3. *Assume $\sim$ is a simulation relation from I, R to J, S. If $x \sim y$ and $y \in J$ is S-well-founded, then $x \in I$ is R-well-founded.*
4. Kleene-Brouwer Theorem. *If $(T, \prec_1)$ is a tree of sequences over `Nat`, and $(T, \prec_1)$ is well-founded, then $(T, \prec\!\!\prec)$ is well-founded.*
5. *A relation R is well-founded if and only if the tree of all R-decreasing sequences is well-founded. $x \in I$ is R-well-founded if and only if the tree of all R-decreasing sequences from x is well-founded.*
6. *Assume $\sim$ is a weak simulation relation from a set I with a relation R to a set J with a relation S. If (J, S) is well-founded and S is transitive then $\sim$ is a simulation.*

Proof.

1. **(a)** *Well-foundedness is inductive.* Assume $\forall y.(yRx \implies y \text{ is } R\text{-well-founded})$, in order to prove that x is R-well-founded, that is, that for any inductive predicate X we have $x \in X$. Since X is inductive, then our thesis follows by proving $\forall z.(zRx \implies z \in X)$. In order to prove it, let z be such that zRx. Then by hypothesis z is R-well-founded, hence, by the assumption that X is inductive, we have $z \in X$, as we wished to show.
   (b) Assume (x is R-well-founded), in order to prove $\forall y.(yRx \implies y \text{ is } R\text{-well-founded})$, that is, that for any inductive predicate X and any $y \in I$ we have $y \in X$. If X is inductive, then we may prove by definition unfolding that $Y = \{x \in I | X(x) \wedge \forall y.(yRx) \implies y \in X\}$ is inductive. Then $x \in Y$ by R-well-foundation of x, and by definition of Y we have $\forall y.(yRx) \implies y \in X$, as wished.
2. **(a)** Assume T is a tree and all proper descendants of the root of T are $\prec_1$-well-founded. We have to prove that all nodes of T are $\prec_1$-well-founded: we only have to prove that the root is well-founded. All children of the root are $\prec_1$-well-founded by assumption, therefore the root is $\prec_1$-well-founded by point 1.*a* above.
   (b) Assume that the root of T is $\prec_1$-well-founded. Then all nodes of T, being reachable from the root, are $\prec_1$-well-founded by point 1.*b* above.
3. Assume J, S is well-founded and $x \sim y$ and y is S-well-founded in order to prove that x is R-well-founded. We prove that all $x \sim y$ are R-well-founded by induction on $y \in J$. By point 1, it is enough to prove that all zRx are R-well-founded. By definition of simulation

there is some $t \in J$ such that $z \sim t$ and tSy. By induction hypothesis on t we conclude that z is R-well-founded, as we wished to show.

4. *Kleene-Brouwer Theorem.* Assume that $(T, \prec_1)$ is well-founded and T is a tree of sequences over $\mathtt{Nat}$. For any $\sigma \in T$, denote with T_σ the set $\{\tau | \sigma * \tau \in T\}$. We prove that T_σ, $\prec\!\!\prec$ is well-founded by $\prec_1$-induction on σ. The thesis will follow by choosing $\sigma = \langle\rangle$: in this case we have $T_\sigma = T$. Assume that $(T_{\sigma'}, \prec\!\!\prec)$ is well-founded for all $\sigma' \prec_1 \sigma$, in order to prove that T_σ, $\prec\!\!\prec$ is well-founded. All $\sigma' \prec_1 \sigma$ have the form $\sigma @ n$ for some $n \in \mathtt{Nat}$. All $\tau \in T_\sigma$ but the root $\langle\rangle$ are $\langle n \rangle * \rho$ for some $n \in \mathtt{Nat}$ and some $\rho \in T_{\sigma @ n}$. Define $J = \{(n, \rho) | n \in \mathtt{Nat} \wedge \rho \in T_{\sigma @ n}\}$. Let $(n', \rho') S (n, \rho)$ if and only if $n' < n$ or $n' = n$ and $\rho' \prec\!\!\prec \rho$. Then J is well-founded: the proof is by principal induction on n, $<$ and secondary induction on $\rho \in T_{\sigma @ n}$, $\prec\!\!\prec$. For all $\tau \in T_\sigma$, we define a relation $\tau \sim (n, \rho)$ if and only if $\tau = \langle n \rangle * \rho$. $\sim$ is a simulation of T_σ, $\prec\!\!\prec$ in J, S. Indeed, if $\tau' \prec\!\!\prec \tau$ and $\tau \sim (n, \rho)$ then $\tau' = \langle n' \rangle * \rho'$. By definition of $\prec\!\!\prec$, either $n' < n$ or $n' = n$ and $\rho' \prec\!\!\prec \rho$. In both cases by definition of $\sim$ and S we have: $\tau' \sim (n', \rho')$ and $(n', \tau') S (n, \tau)$. Thus, by point 3 above τ is $\prec_1$-well-founded, for all $\tau = \langle n \rangle * \rho$, that is, for all $\tau \in T_\sigma$ different from the root $\langle\rangle$. By point 2 we conclude that T_σ is well-founded.
5. Assume T is the tree of all R-decreasing sequences on I (all $\langle x_1, \ldots, x_n \rangle$ such that $x_n R x_{n-1} R \ldots R x_1$). We may define a simulation $\sim$ of (I, R) into $(T, \prec_1)$ by $x \sim \langle x_1, \ldots, x_n \rangle$ if and only if $n > 0$ and $x = x_n$. $\sim$ is a bisimulation. Indeed, assume that $x \sim \langle x_1, \ldots, x_n \rangle$, that is, $n > 0$ and $x = x_n$. If yRx then $\langle x_1, \ldots, x_n, y \rangle$ is R-decreasing and $y \sim \langle x_1, \ldots, x_n, y \rangle$. Conversely, if $\langle x_1, \ldots, x_n, y \rangle$ is R-decreasing then $yRx_n = x$. For all $x \in I$ we have $\langle x \rangle \in T$ and $x \sim \langle x \rangle$. Thus, if T is $\prec_1$-well-founded then all $x \in I$ are R-well-founded by point 3. If R is well-founded then all sequences in T with one or more points are well-founded by 3, therefore $(T, \prec_1)$ is well-founded by point 2 above.
   Assume T_x is the tree of all R-decreasing sequences $\langle x_1, \ldots, x_n \rangle$ on I such that $n > 0$ and $x_n = x$. Define $\sim'$ by restricting $\sim$ to the set of all pairs $(y, \langle x_1, \ldots, x_n \rangle)$ such that $\langle x_1, \ldots, x_n \rangle \in T_x$ (that is, such that $n > 0$ and $x_n = x$ and $x_1 = x$). Then a reasoning similar to the previous one shows that $\sim'$ is a bisimulation, therefore x is R-well-founded if and only if $\langle x \rangle$ is well-founded in T_x. By point 2, since $\langle x \rangle$ is the root of T_x, this is equivalent to the fact that T_x is well-founded.
6. Assume $\sim$ is a weak simulation relation from a set I with a relation R to a set J with a relation S. Assume $x \sim y$ and zRx in order to prove that for some tSy we have $z \sim t$. We argue by induction on y w.r.t. S. By definition of weak simulation, for some tSy we have either $x \sim t$ or $z \sim t$. In the second case we have the thesis. In the first case by induction hypothesis on t there is some uSt such that $z \sim u$. By transitivity of S and uSt, tSy we have uSy: we conclude our thesis. ◀

4 The proof idea for the conservativity result

Assume π is a proof of $\mathtt{PA}^\omega$ or of $\mathtt{PA}^\omega$ + recursive ω-rule. Recall that we write $\pi : \Gamma \vdash \Delta$ for "π has conclusion $\Gamma \vdash \Delta$", and that a proof is cut-free if it includes no cut rule. We consider the notion of sub-formula in which the sub-formulas of $\forall x.A$ (quantification over $\mathtt{Nat}$) are all $A[n/x]$ with $n \in \mathtt{Nat}$. A proofs of $\mathtt{PA}^\omega$ + recursive ω-rule satisfies the sub-formula property if all formulas in any sequent of the proof are a sub-formula of some formula in the conclusion and they occur in the left-hand-side if they are negative sub-formulas, in the right-hand-side if they are positive sub-formulas.

Let $R, R_1, \ldots, R_n$ be primitive recursive binary predicates on $\mathtt{Nat}$. We assume $\mathtt{PA}^\omega \vdash \mathtt{WF}(R)$ in order to intuitionistically derive $\mathtt{WF}(R)$. Then we will generalize the result to a statement

of the form $\mathtt{WF}(R_1), \ldots, \mathtt{WF}(R_n) \implies \mathtt{WF}(R)$. We first recall some well-known intuitionistic results about $\mathtt{PA}^n$ and $\mathtt{PA}^n$ + recursive ω-rule.

Our first step is to prove that if $\mathtt{PA}^n \vdash \mathtt{WF}(R)$, then $\mathtt{PA}^n$ + recursive ω-rule $\vdash \mathtt{WF}(R)$ with some normal proof.

▶ **Lemma 8** (Embedding and Normalization). *Let $\Gamma \vdash \Delta$ be a sequent and π be a proof of $\mathtt{PA}^\omega$. Assume σ be any substitution of the first order free variables of $\Gamma \vdash \Delta$ with numerals.*

1. *There is a recursive map f taking any proof $\pi : \Gamma \vdash \Delta$ in $\mathtt{PA}^\omega$, any first order substitution σ, and returning a proof $\Pi = f(\pi, \sigma) : \sigma(\Gamma \vdash \Delta)$ in $\mathtt{PA}^\omega$ + recursive ω-rule.*
2. *Let $n > 0$. There is a recursive map g, taking any infinitary proof-tree $\pi : \Gamma \vdash \Delta$ in $\mathtt{PA}^n$ + recursive ω-rule, and returning some cut-free proof $\Pi = g(\pi) : \Gamma \vdash \Delta$ in the same system. This fact has an intuitionistic proof using Comprehension of order $n + 1$.*
3. *Any normal proof of $\mathtt{PA}^\omega$ + recursive ω-rule satisfies the sub-formula property.*

Proof.

1. We recursively define a map f taking any proof $\pi : \Gamma \vdash \Delta$ in $\mathtt{PA}^\omega$, any first order substitution σ, and returning a proof $f(\pi, \sigma) : \sigma(\Gamma \vdash \Delta)$ in $\mathtt{PA}^\omega$ + recursive ω-rule. The proof follows the pattern of the analogous result for $\mathtt{PA}^1$ obtained by Tait ([17], p. 277, Thm. 28.9). Any rule of $\mathtt{PA}^\omega$, different from the rule $\forall_r$ for $\mathtt{Nat}$ and from the rule $\mathtt{ind}$, is translated in $\mathtt{PA}^\omega$ + recursive ω-rule by the rule itself . There are two cases left.
   a. Assume π ends with some $\forall_r$-rule, whose unique assumption is $\pi_1 : \Gamma \vdash \Delta, \forall x A, A[z/x]$ in $\mathtt{PA}^\omega$, for some $z \notin \mathtt{FV}(\Gamma \vdash \Delta, \forall x.A)$. Let $\sigma[n/z]$ be any extension of the substitution σ to z with some numeral n. By assumption z is not free in $\Gamma, \Delta, \forall x A$, hence z may occur free in A only if $z = x$. σ is a closed substitution, therefore z occurs in no $\sigma(y)$. Thus, $\sigma[n/z](\Gamma \vdash \Delta, \forall x A) = \sigma(\Gamma \vdash \Delta, \forall x A)$ and $\sigma[n/z](A[z/x]) = \sigma(A[z/x][n/z]) = \sigma(A[n/x]) = \sigma(A)[n/x]$. Then by induction hypothesis we have $f(\pi_1, \sigma[n/z]) : \sigma[n/z](\Gamma \vdash \Delta, \forall x A, A[z/x]) = \sigma(\Gamma \vdash \Delta, \forall x A, A[n/x])$. Eventually, we define some proof $f(\pi, \sigma) : \sigma(\Gamma \vdash \Delta, \forall x.A)$ by recursive ω-rule.
   b. Assume π ends with some $\mathtt{ind}$-rule, whose assumptions are $\pi_1 : \Gamma \vdash \Delta, \forall x A, A[0/x]$ and $\pi_2 : \Gamma, A[z/x] \vdash \Delta, \forall x A, A[z + 1/x]$ in $\mathtt{PA}^\omega$, for some $z \notin \mathtt{FV}(\Gamma \vdash \Delta, \forall x.A)$. Let $\sigma[n/z]$ be any extension of the substitution σ to z with some numeral n. With the same reasoning we did on z in the previous case we obtain $\sigma[n/z](A[0/x]) = \sigma(A)[0/x]$ and $\sigma[n/z](A[z/x]) = \sigma(A)[n/x]$ and $\sigma[n/z](A[z + 1/x]) = \sigma(A)[n + 1/x]$. Then by induction hypothesis we have $f(\pi_1, \sigma[n/z]) : \sigma[n/z](\Gamma \vdash \Delta, \forall x A, A[0/x]) = \sigma(\Gamma \vdash \Delta, \forall x A, A[0/x])$ and $f(\pi_2, \sigma[n/z]) : \sigma[n/z](\Gamma, A[z/x] \vdash \Delta, \forall x A, A[z + 1/x]) = \sigma(\Gamma, A[n/x] \vdash \Delta, \forall x A, A[n + 1/x])$. We inductively define a recursive family $\Pi_n : \sigma(\Gamma \vdash \Delta, \forall x A, A[n/x])$ of proofs indexed by $n \in \mathtt{Nat}$ by $\Pi_0 = f(\pi_1, \sigma[n/z]) : \sigma(\Gamma \vdash \Delta, \forall x A, A[0/x])$ and $\Pi_{n+1} =$ the cut of $\Pi_n : \sigma(\Gamma \vdash \Delta, \forall x A, A[n/x])$ and $f(\pi_2, \sigma[n/z]) : \sigma(\Gamma, A[n/x] \vdash \Delta, \forall x A, A[z + 1/x])$. Eventually, we define $f(\pi, \sigma) : \sigma(\Gamma \vdash \Delta, \forall x.A)$ by recursive ω-rule from $\{\Pi_n | n \in \mathtt{Nat}\}$.
2. (Proof Sketch). Using Girard's candidates, adapted to the sequent calculus. T. Altenkirch formalized Girard's candidates in LEGO using an inductive definition over second order formulas and third order quantifiers ([1], p.109). His idea may be easily generalized: defining Girard's candidates for $\mathtt{PA}^n + \omega$-rule requires Comprehension of order $n + 1$. We need Comprehension of order 3 for defining Girard's candidates for $\mathtt{PA}^2$, and so forth. We postpone the details to a journal version of this paper.
3. By induction over the normal proof. ◀

In order to derive $\mathtt{WF}(R)$, now it is enough to prove that for a normal proof π of $\mathtt{WF}(R)$ in $\mathtt{PA}^\omega$ + recursive ω-rule there is simulation relation $\sim$ between the R-decreasing sequences

with the one-step extension relation, and the proof-tree π itself, with the post-order relation $\prec\!\!\prec$. By Lemma 7.6, even a weak simulation relation is enough. Then our conservativity result will follow by Lemma 7.3. The proof is intuitionistic because Lemma 7 is.

5 Simulating a primitive recursive relation into a normal infinitary proof of its well-foundedness

In this section we will define a simulation relation $\sim$ between the tree of decreasing R-sequences and π, $\prec\!\!\prec$, a normal proof-tree with ω-rule of the statement $\mathtt{WF}(R)$, with the post-order relation.

Let R be the relation we fixed, $I = \mathtt{Nat}$, $x, y \in I$. Assume X is any unary predicate variable. Assume that π is a normal proof in $\mathtt{PA}^\omega$ + recursive ω-rule of $\vdash \mathtt{WF}(R)$.

By the sub-formula property for normal proofs (Lemma 8.3), for any sequent $\Gamma \vdash \Delta$ occurring in π, all $A \in \Gamma$ are negative sub-formulas of $\mathtt{WF}(R)$, and all $B \in \Delta$ are positive sub-formulas of $\mathtt{WF}(R)$. We refer to Def. 4 for the names we assigned to the sub-formulas of $\mathtt{WF}(R)$. The immediate sub-formulas of $\mathtt{WF}(R) = \forall x.\mathtt{Wf}(x)$ are $\mathtt{Wf}(n)$ for all $n \in \mathtt{Nat}$ (positive). The immediate sub-formula of $\mathtt{Wf}(n) = \forall X.\mathtt{wf}(n, X)$ is $\mathtt{wf}(n, X)$ (positive). The sub-formulas of $\mathtt{wf}(n, X) = (\mathtt{IND}(X) \implies X(n))$ are: $X(n)$ (positive) $\mathtt{IND}(X) = \forall y.\mathtt{Ind}(y, X)$ (negative), for all $m \in \mathtt{Nat}$, $\mathtt{Ind}(m, X) = (\mathtt{IH}(m, X) \implies X(m))$ (negative), $X(m)$ (negative), $\mathtt{IH}(m, X) = \forall z.(zRm \implies X(z))$ (positive), for all $p \in \mathtt{Nat}$, $(pRm \implies X(p)$ (positive), $X(p)$ (positive), pRm (negative). All sub-formulas of $\mathtt{WF}(R)$ but those of the form $X(m)$ for some m have a unique sign. Summing up, we just proved:

▶ **Lemma 9** (Sequents in π). *Every formula F in every sequent $\Gamma \vdash \Delta$ in any normal proof π of $\vdash \mathtt{WF}(R)$ in $\mathtt{PA}^\omega$ + recursive ω-rule falls in at least one of the following cases, for some $n, m, p \in \mathtt{Nat}$:*

1. $F = \mathtt{WF}(R) = \forall x.\mathtt{Wf}(x) \in \Delta$
2. $F = \mathtt{Wf}(n) = \forall X.\mathtt{wf}(n, X) \in \Delta$
3. $F = \mathtt{wf}(n, X) = (\mathtt{IND}(X) \implies X(n)) \in \Delta$
4. $F = \mathtt{IND}(X) = \forall y.\mathtt{Ind}(y, X) \in \Gamma$
5. $F = \mathtt{Ind}(m, X) = (\mathtt{IH}(m, X) \implies X(m)) \in \Gamma$
6. *1.* $F = X(m) \in \Gamma$
   2. $F = X(m) \in \Delta$
3. $F = \mathtt{IH}(m, X) = \forall z.(zRm \implies X(z)) \in \Delta$
4. $F = ((pRm) \implies X(p)) \in \Delta$
5. $F = (pRm) \in \Gamma$

We apply the post-order relation $\prec\!\!\prec$ to the proof-tree π, taking the same order among the premises of a rule we have in the proof (this order is fixed in Def. 2).

Let us denote with T the tree of R-decreasing sequences on $\mathtt{Nat}$. We will define a weak simulation relation of $(T, \prec_1)$ in $(\pi, \prec\!\!\prec)$, relating any node of T with some node in π. From the well-foundedness of π, $\prec\!\!\prec$ (Lemma 7.4) and the fact that $\prec\!\!\prec$ is transitive we will deduce that $\sim$ is a simulation (Lemma 7.6), therefore any node of T is well-founded (Lemma 7.3). We will conclude that T is well-founded, and R itself is well-founded (Lemma 7.5).

If ν is any node of π, we write $\nu : \Gamma \vdash \Delta$ for "the sub-proof of π of root ν has conclusion $\Gamma \vdash \Delta$". Let $\sigma = a_k R a_{k-1} R \dots R a_2 R a_1 \in T$ be any R-decreasing chain. We define a relation $\sigma \sim \nu$ between R-sequences and nodes of π.

Informally, $\sigma \sim \nu : \Gamma \vdash \Delta$ says that Δ is a property of the nodes $a_k, \dots, a_1$ of the R-decreasing sequence σ, and in all formulas $aRb \in \Gamma$ and $aRb \implies X(a) \in \Delta$, the element a is the successor of b in σ. The precise definition follows.

▶ **Definition 10** (The relation $\sim$). Let T be the tree of R-decreasing sequences and $\sigma = a_k R a_{k-1} R \dots R a_2 R a_1 \in T$ be any R-decreasing chain, with possibly $k = 0$. Let $\pi : \vdash \mathtt{WF}(R)$ be a normal proof in $\mathtt{PA}^\omega + \omega$-rule. We say that σ is simulated by a node $\nu : \Gamma \vdash \Delta$ in π, and we write $\sigma \sim \nu$, if:

1. every numeral m occurring in Δ is a_i for some $i = 1, \dots, k$
2. every (closed) formula $(pRm) \in \Gamma$ and $(pRm) \implies X(p) \in \Delta$ is, respectively, $(a_{i+1} R a_i)$, $(a_{i+1} R a_i) \implies X(a_{i+1})$ for some $i = 1, \dots, k-1$

ν is a node of π, therefore Lemma 9 lists all formulas which may occur in the conclusion $\Gamma \vdash \Delta$ of ν. Using Lemma 9 we may reformulate the definition of $\sim$ in the following equivalent form.

▶ **Lemma 11** (An alternative definition of $\sim$). *$\sigma \sim \nu : \Gamma \vdash \Delta$ if and only if every formula $F \in \Gamma \vdash \Delta$ falls in at least one of the following cases:*

1. $F = \mathtt{WF}(R) = \forall x.\mathtt{Wf}(x) \in \Delta$
2. $F = \mathtt{Wf}(a_i) = \forall X.\mathtt{wf}(a_i, X) \in \Delta$, *for some* $a_i \in \sigma$
3. $F = \mathtt{wf}(a_i, X) = (\mathtt{IND}(X) \implies X(a_i)) \in \Delta$, *for some* $a_i \in \sigma$
4. $F = \mathtt{IND}(X) \in \Gamma$
5. $F = \mathtt{Ind}(m, X) \in \Gamma$ *for some* $m \in \mathtt{Nat}$
6. *1.* $F = X(m) \in \Gamma$ *for some* $m \in \mathtt{Nat}$
   2. $F = X(a_i) \in \Delta$ *for some* $a_i \in \sigma$.
3. $F = \mathtt{IH}(a_i, X) \in \Delta$ *for some* $a_i \in \sigma$.
4. $F = ((a_{i+1} R a_i) \implies X(a_{i+1})) \in \Delta$ *for some* $a_i, a_{i+1} \in \sigma$.
5. $F = (a_{i+1} R a_i) \in \Gamma$ *for some* $a_i, a_{i+1} \in \sigma$.

The relation $\sim$ is closed under ancestor: if $\sigma \sim \nu$ and $\nu \prec \mu$ then $\sigma \sim \mu$. The reason is that all rules $\neq \mathtt{cut}$ of $\mathtt{PA}^\omega$ + recursive ω-rule are contravariant w.r.t. the descendant relation $\prec$: if $\nu : \Gamma \vdash \Delta$, $\nu' : \Gamma' \vdash \Delta'$ and $\nu \prec \nu'$ then $\Gamma \vdash \Delta \supseteq \Gamma' \vdash \Delta'$, therefore $\nu' : \Gamma' \vdash \Delta'$ satisfies the clauses for σ if $\nu : \Gamma \vdash \Delta$ satisfies the clauses for σ.

The relation $\sim$ is total: for every $\sigma \in T$ there is some $\nu \in \pi$ such that $\sigma \sim \nu$. *Proof.* Let μ_0 be the root of π. Then we have $\mu_0 : \vdash \mathtt{WF}(R)$. Every F in $\vdash \mathtt{WF}(R)$ satisfies $F = \mathtt{WF}(R) \in \Delta$. By Lemma 11 we have $\sigma \sim \mu_0$ for every $\sigma \in T$.

We recall that $(\tau \prec_1 \sigma)$ means that τ is of the form $a_{k+1} R a_k R \dots R a_1$, that is, that τ is a generic one-step extension of σ. We write $\mu \prec_1 \nu$ for "μ is a child of ν in π", too. We first prove that $\sim$ is a weak simulation, then that $\sim$ is a simulation.

▶ **Lemma 12** (Weak simulation). *Let T be the tree of R-decreasing sequences, $\pi : \vdash \mathtt{WF}(R)$ a normal proof in $\mathtt{PA}^\omega + \omega$-rule, and $\sim$ as in Def. 10. Then $\sim$ is a weak simulation between $(T, \prec_1)$ and $(\pi, \prec)$.*

Proof. Assume $\sigma = a_k R a_{k-1} R \dots R a_1$ and $\tau = a_{k+1} R a_k R \dots R a_1 \prec_1 \sigma$ and $\sigma \sim \nu$. We have to prove that there is some $\mu \prec_1 \nu$ in π such that $\sigma \sim \mu$, or $\tau \sim \mu$. There are 9 classes of formulas $\in \Gamma \vdash \Delta$. We distinguish 9 cases according to the formula inferred in ν.

1. $\mathtt{WF}(R)$. We infer $\mathtt{WF}(R) \in \Delta$ from all $\mathtt{Wf}(p) \in \Delta$ with $p \in \mathtt{Nat}$, using the ω-rule with one premise $\mu_p \prec_1 \nu$ for each p. We choose $p = a_{k+1}$, the node added in τ. The extra formula $\mathtt{Wf}(a_{k+1})$ we have in the right-hand-side of the premise μ_p satisfies the clause 2 of Lemma 11 for τ, therefore $\tau \sim \mu_p$.
2. $\mathtt{Wf}(a_i)$. We infer $\mathtt{Wf}(a_i) \in \Delta$ from some $\mathtt{wf}(a_i, X) \in \Delta$ using the rule $\forall_r$ with a single premise $\mu \prec_1 \nu$. The extra formula $\mathtt{wf}(a_i, X)$ we have in the right-hand-side of the premise satisfies the clause 3 of Lemma 11 for σ, therefore $\sigma \sim \mu$.

3. $\mathtt{wf}(a_i, X)$ for some $a_i \in \sigma$. We infer $\mathtt{wf}(a_i, X) = (\mathtt{IND}(X) \implies X(a_i)) \in \Delta$ from some $\mathtt{IND}(X)$ and some $X(a_i)$ in the left- and right-hand-side using the rule $\Longrightarrow_r$ with a single premise $\mu \prec_1 \nu$. The extra formulas $\mathtt{IND}(X) \in \Gamma$ and $X(a_i) \in \Delta$ we have in the premise satisfy the clauses 4, 5 of Lemma 11, therefore $\sigma \sim \mu$.
4. $\mathtt{IND}(X)$. We infer $\mathtt{IND}(X) \in \Gamma$ from some $\mathtt{Ind}(m, X) \in \Gamma$ using the rule $\forall_l$ with a single premise $\mu \prec_1 \nu$. The extra formula $\mathtt{Ind}(m, X)$ we have in the left-hand-side of the premise satisfies the clause 5 of Lemma 11, therefore $\sigma \sim \mu$.
5. $\mathtt{Ind}(m, X)$. We infer $\mathtt{Ind}(m, X) = (\mathtt{IH}(m, X) \implies X(m)) \in \Gamma$ using the rule $\Longrightarrow_l$ from two premises: a left premise μ having $\mathtt{IH}(m, X)$ added to Δ, and a right premise μ' having $X(m)$ added to Γ, for some $\mu \prec_1 \mu' \prec_1 \nu$. The formula $X(m)$ added to Γ satisfies the clause 6a of Lemma 11, therefore $\sigma \sim \mu' \prec_1 \nu$.
6. $X(m)$. Using the Identity rule, we infer the same formula $X(m) \in \Gamma$ and $X(m) \in \Delta$, for some $m \in \mathtt{Nat}$. From $X(m) \in \Delta$ we get $m = a_i$ for some $i = 1, \ldots, k$ by the clause 6b of Lemma 11. $X(a_i)$ does not belong to the left-hand-side of the root of π, therefore we may find the last node $\nu \prec \mu$ in the path from ν to the root of π such that $X(a_i)$ belongs to the left-hand-side of the conclusion of μ. The only formula in π which we may prove from some $X(a_i)$ in the left-hand-side is a left occurrence of $\mathtt{Ind}(a_i, X) = (\mathtt{IH}(a_i, X) \implies X(a_i))$. Therefore there is some node η having μ as right premise, in which we introduce by $\Longrightarrow_l$ some $\mathtt{Ind}(a_i, X) = (\mathtt{IH}(a_i, X) \implies X(a_i))$. The left premise of η is some node θ in which $\mathtt{IH}(a_i, X)$ occurs in the right-hand-side and it is used by $\Longrightarrow_l$ to derive $\mathtt{Ind}(a_i, X)$. We sum up the situation in the proof tree below.

$$\dfrac{\theta : \Gamma'', \mathtt{Ind}(a_i, X) \vdash \mathtt{IH}(a_i, X), \Delta'' \qquad \begin{array}{c} \dfrac{}{\nu : \Gamma', X(a_i) \vdash X(a_i), \Delta'}\ \mathtt{id} \\ \vdots \\ \mu : \Gamma'', \mathtt{Ind}(a_i, X), X(a_i) \vdash \Delta'' \end{array}}{\eta : \Gamma'', \mathtt{Ind}(a_i, X) \vdash \Delta''}\ \Longrightarrow_l$$

   We have $\sigma \sim \eta$ because η is an ancestor of ν. $\mathtt{IH}(a_i, X)$ satisfies the clause 7 of Lemma 11, therefore $\sigma \sim \theta$. We have $\theta \lessdot \nu$ because θ is the left premise and μ the right premise of η, and μ is an ancestor of ν.
7. $\mathtt{IH}(a_i, X)$. For some $i = 0, \ldots, k$, we infer $\mathtt{IH}(a_i, X) \in \Delta$ using the recursive ω-rule. For every $p \in \mathtt{Nat}$, there is some assumption μ_p of ν adding to Δ the formula $F_p = (pRa_i \implies X(p))$. In post-order we have $\mu_0 \lessdot \mu_1 \lessdot \mu_2 \lessdot \ldots \lessdot \nu$. We distinguish two sub-cases according to $i < k$ or $i = k$.

   a. Let $i < k$. Then $i + 1 \leq k$, therefore $(a_{i+1}Ra_i \implies X(a_{i+1}))$ has the form required by clause 8 of Lemma 11 for σ. If we choose $p = a_{i+1}$, we conclude $\sigma \sim \mu_p$, for some $\mu_p \prec_1 \nu$.
   b. Let $i = k$. Then a_{k+1} is the last element of τ. The formula $(a_{k+1}Ra_k) \implies X(a_{k+1})$ has the form required by clause 8 of Lemma 11 for τ. If we choose $p = a_{i+1}$, we conclude $\tau \sim \mu_p$ for some $\mu_p \prec_1 \nu$.

8. $(a_{i+1}Ra_i \implies X(a_i), \Longrightarrow_r)$. We infer some formula $a_{i+1}Ra_i \implies X(a_{i+1}) \in \Delta$ using the rule $\Longrightarrow_r$, from one premise μ having $(a_{i+1}Ra_i)$ in the left-hand-side and $X(a_i)$ in the right-hand-side. The first formula satisfies the clause 9 of Lemma 11, the second one the clause 6b of the same Lemma. We conclude $\sigma \sim \mu \prec_1 \nu$.
9. *One or more formulas aRb.* We infer some atomic formulas using the rule `axiom` and one basic arithmetical axiom $\alpha_1, \ldots, \alpha_n \vdash \alpha$ or $\beta_1, \ldots, \beta_n \vdash$, with all α_i, α of the form $p(t_1, \ldots, t_m)$ for some $p, t_1, \ldots, t_m$. All atomic formulas $p(t_1, \ldots, t_m)$ in π have the form

$a_{i+1}Ra_i$, are closed, are true by assumption on σ, and are in the left-hand side. Thus, the basic arithmetical axiom has the form $\alpha_1, \ldots, \alpha_n \vdash$, with all α_i closed and true. *This case cannot happen*, because `axiom` may infer a closed sequence $\alpha_1, \ldots, \alpha_n \vdash$ only if it is true, hence if some α_i is false. ◀

We have now all the ingredients we need to intuitionistically derive our conservativity result.

▶ **Lemma 13** (well-foundedness of R). *Assume R is any primitive recursive binary relation on* `Nat`*, and $(T, \prec_1)$ is the tree of R-decreasing sequences with the child/father relation. Let $\pi : \mathtt{WF}(R)$ be any proof of $\mathtt{WF}(R)$ in $\mathtt{PA}^\omega$, and $\Pi = gf(\pi) : \mathtt{WF}(R)$ be the normal proof of $\mathtt{WF}(R)$ in $\mathtt{PA}^\omega +$ recursive ω-rule, obtained by 8.1, 2. Let $\sim$ be the simulation relation defined above, and $\prec\!\!\prec$ the post-order ordering on Π.*

1. *Π with $\prec\!\!\prec$ is well-founded.*
2. *$\sim$ is a simulation relation of $(T, \prec_1)$ in Π, $\prec\!\!\prec$.*
3. *$(T, \prec_1)$ is well founded*
4. *R is well founded.*

Proof.

1. By the Kleene-Brouwer Theorem (Lemma 7.4) and the hypothesis that Π, $\prec_1$ is well-founded.
2. By Lemma 7.6, the fact that $\sim$ is a weak simulation (Lemma 12), and that $(\Pi, \prec\!\!\prec)$ is well-founded (point 1).
3. By point 2 above, $\sim$ is a simulation. By point 1, the root of Π is $\prec\!\!\prec$-well-founded. Any node of T is related to the root of Π by $\sim$, hence by Lemma 7.3, any node of T is $\prec_1$-well-founded. We conclude that $(T, \prec_1)$ is well-founded.
4. R is well founded by Lemma 7.5, because $(T, \prec_1)$ is well-founded by point 3. ◀

Eventually, we conclude:

▶ **Theorem 14** (Conservativity). *Assume $R, R_1, \ldots, R_n$ are any binary primitive recursive relation over* `Nat` *and $n > 0$.*

1. *If $\mathtt{PA}^n \vdash \mathtt{WF}(R_1), \ldots, \mathtt{WF}(R_n) \implies \mathtt{WF}(R)$, then using Comprehension of order $n+1$ we may intuitionistically derive $\mathtt{WF}(R_1), \ldots, \mathtt{WF}(R_n) \implies \mathtt{WF}(R)$.*
2. *If $\mathtt{PA}^\omega \vdash \mathtt{WF}(R_1), \ldots, \mathtt{WF}(R_n) \implies \mathtt{WF}(R)$, then using Comprehension of order ω we may intuitionistically derive $\mathtt{WF}(R_1), \ldots, \mathtt{WF}(R_n) \implies \mathtt{WF}(R)$.*

Proof.

1. Assume $\mathtt{WF}(R_1)$, …, $\mathtt{WF}(R_n)$ and $\mathtt{PA}^n \vdash \mathtt{WF}(R_1), \ldots, \mathtt{WF}(R_n) \implies \mathtt{WF}(R)$, in order to deduce $\mathtt{WF}(R)$. By Lemma 5 we have $\mathtt{PA}^n \vdash \mathtt{WF}(R_1)$ and … and $\mathtt{PA}^n \vdash \mathtt{WF}(R_n)$. By cut rule we deduce $\mathtt{PA}^n \vdash \mathtt{WF}(R)$. By Lemma 13.4 we conclude $\mathtt{WF}(R)$. The proof is intuitionistic because all proofs in this paper are intuitionistic. The proof may be obtained using comprehension of order $n+1$ because this is the case for normalization for $\mathtt{PA}^n + \omega$-rule.
2. By the previous point. ◀

6 Conclusions

The aim of this work is to isolate classes $\mathcal{C}$ of formulas such that all proofs of formulas in $\mathcal{C}$ may be turned into intuitionistic proofs, at least in principle. The idea is that, if we care about proving a mathematical result having a concrete meaning, and we have a theorem which is classically but not intuitionistically provable, then we should reformulate our goal

in order to obtain one formula in one of these classes. Then we may prove our result freely using classical logic, knowing that, afterward, our proof may always be made intuitionistic. The advantage is that it is much easier to restrict ourselves to a goal in a class $\mathcal{C}$ of formulas, instead than to the use of intuitionistic logic in the proof of the goal. Intuitionistic proofs provide extra information, but if we choose to prove a statement in $\mathcal{C}$, then we know that the intuitionistic proof may always be done as a second step. As a first step we check whether the statement is classically true, a much easier task.

The very first example for $\mathcal{C}$ was provided by K. Gödel. In this case, $\mathcal{C}$ is the set of formulas obtained inserting a double negation ($\neg\neg$) everywhere. However, this class $\mathcal{C}$ has mainly an interest from a foundational viewpoint, because in intuitionistic logic the proof of a negation provides no concrete information.

A second example for $\mathcal{C}$ is the set of formulas provided by the Dialectica interpretation. In this case we have formulas whose intuitionistic proofs are rich of concrete consequences. Indeed, these formulas are successfully used by U. Kohlenbach [11] and others to analyze, say, proofs of fixed point results for non-expansive maps. A drawback of this class, however, lies in the complexity of these formulas, which are long, involved, and use functionals of higher types. Often, formulas in $\mathcal{C}$ require a real effort to be understood.

Another choice for $\mathcal{C}$ is the set of Π^0_2-sentences, those of the form $\forall x \in \mathtt{Nat}.\exists y \in \mathtt{Nat}.R(x,y)$, for any primitive recursive binary relation R on $\mathtt{Nat}$. The conservativity result for this class was proved by G. Kreisel [12]. Later, H. Friedman provided A-translation, the first realistic and purely mechanical method for turning a classical proof of a Π^0_2-formula into an intuitionistic one of the same formula [8]. An intuitionistic proof of a Π^0_2-formula is not a mere existence result, but it outlines a method for computing y given x. The only limitation is that this class is very narrow: in a proof of a Π^0_2-formula we often use as lemmas some statements which are much more complex in the arithmetical hierarchy. We would like intuitionistically provable versions for lemmas, too.

One way to overcome this limitation is to generalize A-translation to a larger class of first order formulas. Berger, Buchholz and Schwichtenberg [4] proved, for instance, that we may take as $\mathcal{C}$ the set of all sequents $\Gamma \vdash A$ with A some Π^0_2-formula and Γ containing only formulas $\forall x_1, \ldots, x_n.(\alpha_1, \ldots, \alpha_m \implies \alpha)$, with α and all α_i atomic and different from the constant $\bot$ (false). There are more results along this line [4] .

However, there are Π^0_2-theorems requiring more than first order formulas. For instance, Paris-Harrington theorem, the Theorem about Goodstein sequences, Higman Lemma and Kruskal Lemma are Π^0_2-theorems whose proof requires second order formulas. Therefore it makes sense to look for a choice of $\mathcal{C}$ including some second order formulas.

In this paper we considered a choice of this kind, all statements of the form: "if $\mathtt{WF}(R_1)$, $\ldots$, $\mathtt{WF}(R_n)$ then $\mathtt{WF}(R)$", where $\mathtt{WF}(S)$ means: "S is inductively well-founded", again for any primitive recursive binary relations S on $\mathtt{Nat}$.

6.1 Some corollaries of the conservativity result

The conservativity results holds for statements of the form: "$\mathtt{WF}(R_1), \ldots, \mathtt{WF}(R_n) \implies \mathtt{WF}(R)$". The relevance of this class relies on the empirical evidence that many mathematical results, if expressed in this form, are rich of concrete and interesting intuitionistic consequences.

For instance, *Higman's Lemma* may be expressed in the form $\mathtt{WF}(R)$ for some primitive recursive R, as follows. Let us denote with $\mathtt{List}(I)$ the set of finite lists over some set I. For any $L, M \in \mathtt{List}(I)$, we write $L \sqsubseteq M$ if L may be obtained from M by possibly skipping some elements of M. Let T be the set of $\langle L_1, \ldots, L_n\rangle \in \mathtt{List}(\mathtt{List}(\mathtt{Nat}))$ such that for all $0 < i < j < n$ we have $L_i \not\sqsubseteq L_j$. Then Higman Lemma may be expressed by

saying that the set T is well-founded by one-step extension (see [5], [7]). A similar remark applies to Kruskal's Lemma. More in general, the theory of Almost Full Relations, the intuitionistic version of Quasi-Well-Orders, may be developed using theorems of the form $\mathtt{WF}(R_1), \ldots, \mathtt{WF}(R_n) \implies \mathtt{WF}(R)$ ([20]).

Another example is *Ramsey's Theorem*. Let $G \subseteq \mathtt{Nat}$ be any primitive recursive complete graph and $c : G \times G \to \{1, \ldots, n\}$ be any primitive recursive n-color assignment . Define, for $i = 1, \ldots, n$, T_i as the set of all finite increasing lists $\langle x_1, \ldots, x_n \rangle$ over $\mathtt{Nat}$, such that for all $0 < j < k < n$ we have $x_j, x_k \in G$ and $c(x_j, x_k) = i$. Define T as the set of all finite increasing lists $\langle x_1, \ldots, x_n \rangle$ over $\mathtt{Nat}$, such that for all $0 < j < k < n$ we have $x_j, x_k \in G$. Then we may express Ramsey theorem by saying: if $T_1, \ldots, T_n$ are well-founded then T is well-founded. Indeed, by definition unfolding, this implication means: if, for $i = 1, \ldots, n$, the tree of finite i-colored sub-graph of G is well-founded by one-point extension, then the tree of all finite sub-graphs of G is well-founded by one-step extension. This latter is just a way of saying: G itself is finite. We recognize this statement as a variant of the contrapositive of Ramsey. By Theorem 14 this statement has an intuitionistic proof, whenever G, c are primitive recursive. In fact, this variant of Ramsey theorem has an intuitionistic proof for *any* G, c ([3]), but Theorem 14 cannot prove it.

Besides, an intuitionistic proof of "R is inductively well-founded" is not just a proof of: "there is some bound to the ordinal height of R", it effectively provides such a bound. This extra information about ordinals has been used, for instance, to characterize the programs proved terminating by the Termination algorithm [2].

It is worth to quote that the statements of the form $\mathtt{WF}(R)$ are used to develop formal topology ([6]).

Acknowledgments. We thank F. Aschieri, U. Berger, T. Coquand, P. Martin-Löf and P. Oliva for the fruitful suggestions about an earlier version of this paper.

References

1 T. Altenkirch. *Constructions, Inductive Types and Strong Normalization.* PhD thesis, University of Edinburgh, 1993.

2 Stefano Berardi, Paulo Oliva, and Silvia Steila. Proving termination of programs having transition invariants of height ω. In *Proceedings of the 15th Italian Conference on Theoretical Computer Science, Perugia, Italy, September 17-19, 2014.*, pages 237–240, 2014.

3 Stefano Berardi and Silvia Steila. Ramsey theorem as an intuitionistic property of well founded relations. In *Rewriting and Typed Lambda Calculi – Joint International Conference, RTA-TLCA 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, pages 93–107, 2014.

4 Ulrich Berger, Wilfried Buchholz, and Helmut Schwichtenberg. Refined program extraction form classical proofs. *Ann. Pure Appl. Logic*, 114(1-3):3–25, 2002.

5 T. Coquand and D. Fridlender. A proof of higman's lemma by structural induction. unpublished draft.

6 Thierry Coquand, Giovanni Sambin, Jan M. Smith, and Silvio Valentini. Inductively generated formal topologies. *Ann. Pure Appl. Logic*, 124(1-3):71–106, 2003.

7 D. Fridlender. *Higman's lemma in Type Theory.* PhD thesis, Chalmers University, 1997.

8 H. Friedman. Classically and intuitionistically provably recursive functions. *Lecture Notes in Mathematics*, 669:21–27, 1978.

9 J.-Y. Girard. *Proof theory and logical complexity.* Studies in Proof Theory. Monographs, 1. Bibliopolis, 1987.

10 William A. Howard and Georg Kreisel. Transfinite induction and bar induction of types zero and one, and the role of continuity in intuitionistic analysis. *J. Symb. Log.*, 31(3):325–358, 1966.

11 U. Kohlenbach. *Applied Proof Theory: Proof Interpretation and their Use in Mathematics*. Springer-Verlag Berlin Heidelberg, 2008.

12 Georg Kreisel. On the interpretation of non-finitist proofs: Part II. interpretation of number theory. applications. *J. Symb. Log.*, 17(1):43–58, 1952.

13 P. Martin-Lof. Hauptsatz for the intuitionistic theory of iterated inductive definitions. In *Proceedings of the Second Scandinavian Logic Symposium, Oslo*, pages 179–216, 1971.

14 P. Martin-Lof. Hauptsatz for the theory of species. In *Proceedings of the Second Scandinavian Logic Symposium, Oslo*, pages 217–233, 1971.

15 David Michael Ritchie Park. Concurrency and automata on infinite sequences. In *Theoretical Computer Science, 5th GI-Conference, Karlsruhe, Germany, March 23-25, 1981, Proceedings*, pages 167–183, 1981.

16 D. Prawitz. Ideas and results in proof theory. In *Proceedings of the Second Scandinavian Logic Symposium, Oslo*, pages 235–307, 1971.

17 K. Schutte. *Proof Theory*. Springer-Verlag, Berlin Heildelberg New York, 1977.

18 W. Sieg, W. Buchholz, S. Feferman, and W. Pohlers. *Iterated Inductive Definitions and Subsystems of Analysis – Recent Proof Theoretical Studies*, volume 897. Springer Lecture Notes in Mathematics (LNM) Berlin-Heidelberg-New York, 1981.

19 Jean van Heijenoort. *From Frege to Gödel, A Source Book in Mathematical Logic, 1879-1931*. Harward University Press, 2002.

20 Dimitrios Vytiniotis, Thierry Coquand, and David Wahlstedt. Stop when you are almost-full – adventures in constructive termination. In *Interactive Theorem Proving – Third International Conference, ITP 2012, Princeton, NJ, USA, August 13-15, 2012. Proceedings*, pages 250–265, 2012.

Functions out of Higher Truncations

Paolo Capriotti[1], Nicolai Kraus*[1], and Andrea Vezzosi[2]

1 **University of Nottingham, United Kingdom**
{ngk,pvc}@cs.nott.ac.uk
2 **Chalmers University of Technology, Sweden**
vezzosi@chalmers.se

Abstract

In homotopy type theory, the truncation operator $\|-\|_n$ (for a number $n \geq -1$) is often useful if one does not care about the higher structure of a type and wants to avoid coherence problems. However, its elimination principle only allows to eliminate into n-types, which makes it hard to construct functions $\|A\|_n \to B$ if B is not an n-type. This makes it desirable to derive more powerful elimination theorems. We show a first general result: If B is an $(n+1)$-type, then functions $\|A\|_n \to B$ correspond exactly to functions $A \to B$ which are constant on all $(n+1)$-st loop spaces. We give one "elementary" proof and one proof that uses a higher inductive type, both of which require some effort. As a sample application of our result, we show that we can construct "set-based" representations of 1-types, as long as they have "braided" loop spaces. The main result with one of its proofs and the application have been formalised in Agda.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases homotopy type theory, truncation elimination, constancy on loop spaces

Digital Object Identifier 10.4230/LIPIcs.CSL.2015.359

1 Introduction

As it is very well-known, the type constructor Σ of Martin-Löf type theory expresses a very strong form of existence. Although a type of the form $\Sigma\,(a : A)\,.\,P(a)$ is read as "there exists an element in A for which the predicate P holds" under the *propositions as types* view, an element of such a type is more than a proof of mere existence: it includes a very concrete example of an element $a : A$. This is not always satisfying as, for example, the set-theoretic axiom of choice becomes a tautology when translated naively to type theory. The idea of adding a construction which allows to formulate existence in a weaker sense has been studied intensively in various different settings. As far as we know, the first documented appearance are *squash types* in the extensional theory of NuPRL [7]. Later, Awodey and Bauer introduced a similar concept in extensional Martin-Löf type theory, called *bracket types* [4]. Homotopy type theory has introduced the *propositional truncation* operation, written $\|-\|_{-1}$ or simply $\|-\|$ [14]. It forces all elements to be equal, in the sense that the identity type $x = y$ is inhabited for any $x, y : \|A\|_{-1}$, and it is well-known that $x = y$ will in fact be uniquely inhabited (i.e. equivalent, or isomorphic, to the unit type). Classically, $\|A\|_{-1}$ is always equivalent to either the unit type or the empty type, but this is of course not the case in a constructive setting.

* The work of Nicolai Kraus was supported by the Engineering and Physical Sciences Research Council (EPSRC), grant reference EP/M016994/1.

The homotopical view has suggested that propositional truncation is only one out of infinitely many operations that reduce the complexity of a type. As "types are weak ω-groupoids" ([12] and [15]), it is easy to imagine that there is, for every number $n \geq -1$, an operation which trivialises all the structure above level $(n+1)$. In other words, this is a reflector for the category of weak n-groupoids, viewed as a subcategory of weak ω-groupoids, roughly speaking. In homotopy type theory, we write this operation as $\|-\|_n$ ("n-truncation"), and it can be seen and implemented as a *higher inductive type* [14]. The truncation operator $\|-\|_n$ is a *monad* in some appropriate sense (and even a *modality* in the sense of [14]), and if we want to, we can choose to work completely in that monad. Types that are canonically equivalent to their n-truncation are called *n-types*, or *n-truncated types*.

Considering n-types (for some given n) instead of all types is useful if we do not care about or want to avoid potential higher equality proofs. For example, if we formalise algebraic structures such as groups, we may require that the type of group elements is of truncation level 0 in order to match the set-theoretic definition: equality of group elements should be a mere proposition and not carry additional information, that is, there is at most one proof that given group elements are equal. As a consequence, for any type A with an element $a : A$, the type $a = a$ is not necessarily a group. It does have a neutral element an elements can be inverted and composed, corresponding to the fact that equality is reflexive, symmetric, and transitive. However, $a = a$ is not a 0-truncated type. We can use 0-truncation to make up for this, and $\|a = a\|_0$ is indeed a group, called the *fundamental group* of A at basepoint a, while $a = a$ (as *pointed type* also written $\Omega(A, a)$) is the *loop space* at point a.

A drawback of truncations is that it can be hard to get out of them, that is, "to leave the monad". A priori we have, for any type A and number $n \geq -1$, a map $|-| : A \to \|A\|_n$, but there is in general no function in the other direction. The universal property of $\|-\|_n$ says that, via composition with $|-|$, the type of functions $\|A\|_n \to B$ is equivalent to the type $A \to B$, but only if B is n-truncated. To continue with the previous example, an element of the fundamental group of A at basepoint a is really an equivalence class of equality proofs (or *paths*) between a and itself, but it is in general impossible to get a specific representative from such a class; that is, we cannot construct a section of the map $|-| : (a = a) \to \|a = a\|_0$. Of course, we would not have expected anything else: it is unreasonable to assume that we can make this sort of choice without any further assumptions. Although the truncation operator $\|-\|_n$ is often described as "cutting of" higher structure of a type, it is more accurate to think of it as "filling non-trivial loops", which makes it plausible that it is *harder* instead of *easier* to define a function out of $\|A\|_n$ than out of A.

Unlike in the example above, it is in some cases reasonable to expect that we can get a function $\|A\|_n \to B$ even if B is not an n-type. If $\|A\|_{-1}$ tells us that A has some element without revealing a concrete one to us, then a function $\|A\|_{-1} \to B$ should be the same as a function $f : A \to B$ which cannot look at the "input".[1] What exactly this means is difficult to state in general (see [8]), so let us restrict ourselves to the case that B is 0-truncated (also called a *set*). In this case, the statement that "f does not look at its input" can be expressed by saying that f maps any pair of inputs to equal values, $\Pi_{x,y:A}(f(x) = f(y))$. Indeed, it has been shown that a function f with this behaviour gives rise to a map $\|A\|_{-1} \to B$ [10].

Even if we have a function $A \to B$, it can be very hard to tell whether it is possible to construct a function $\|A\|_n \to B$ unless B is an n-type, and if it is possible, there is no direct

[1] This only makes sense if stated internally. Of course, a concrete implementation of f *can* compute differently if applied to different terms of type A. As long as we stay inside the theory, we cannot talk about judgmental equality.

way to do so as the universal property (or the elimination principle) cannot be applied directly. The usual workaround is looking for an n-type C "in the middle", that is such that one has functions $A \to C$ and $C \to B$. One can then apply the elimination principle to construct a function $\|A\|_n \to C$ which, by composition, yields a function $\|A\|_n \to B$ as desired. The type C is constructed ad-hoc, and it is natural to ask for a more powerful elimination principle (or universal property) of $\|-\|_n$ which allows the construction of functions $\|A\|_n \to B$ in a more principled and streamlined way.

This has been done for the (-1)-truncation in previous work [8], where it is shown that functions $\|A\|_{-1} \to B$ correspond exactly to functions $A \to B$ with an infinite tower of coherence conditions. This can be understood as a generalised version of the usual universal property of $\|-\|_{-1}$. If B is known to be n-truncated for some fixed finite n, the infinite tower becomes finite and can be expressed directly in type theory, whereas the existence of *Reedy limits* [13] is necessary for the general case. If B is a 0-type, the "tower" of coherence condition is exactly the single condition $\Pi_{x,y:A}(f(x) = f(y))$ discussed above. If B is even a (-1)-type itself, the tower vanishes completely and the usual universal property remains. Unfortunately, it seems that there is no immediate generalisation of the proof of [8] to n-truncations.

In this paper, we do consider n-truncations for general n, but we assume that B is $(n+1)$-truncated, and already this case seems to be involved. We show that functions $\|A\|_n \to B$ correspond exactly to those functions $A \to B$ that are constant on all $(n+1)$-st loop spaces. We offer two proofs for this fact, one which works in "plain" homotopy type theory with general truncations, and the other involving a higher inductive type. The first proof, which we call the "elementary proof", is close to not even requiring the univalence axiom (the central concept of homotopy type theory expressing that equality in the universe is given by type equivalence). The only reason why univalence is necessary is that we need to be able to translate between truncations ($\|a =_A b\|_n$ is equivalent to $|a| =_{\|A\|_{n+1}} |b|$). The second proof (Section 4) uses an argument that makes crucial use of both a higher inductive type and the univalence axiom, and we therefore call it the "HIT proof". In the HIT proof, we will construct a higher inductive type in such a way that it is the "initial" type through which functions $f : A \to B$ with the property (10) factor, and we will show that this type *is* really $\|A\|_n$. Although we show an equivalence of types, we believe that the main application is the construction of functions $\|A\|_n \to B$, that is, one may often want to use only one direction of the equivalence. Therefore, the result can be used as an elimination principle that is more powerful than the usual recursion principle of the truncation. We also present a sample application (a translation of types into "set-based representation"), and conclude with a discussion on how the generalised statement should look like, and under which assumptions it should be provable.

The main contents of this paper have, in slightly different form, appeared in the second-named author's Ph.D. thesis [9].

Outline. We start by stating the result of the paper in Section 2, and discuss two special cases ($n \equiv -1$ and $n \equiv 0$). In Section 3, we give the "elementary" proof of this result, and in Section 4, the (technically harder, but conceptually clear) proof that uses a higher inductive type. We discuss a sample application of the case $n \equiv 0$ in Section 5, namely a construction of a *set-based representation* of any given type, provided that it fulfils a property that e.g. loop spaces do. Finally, in Section 6, we compare the two proofs with each other. We also compare our result with the *general universal property of the propositional truncation* as proved before [8], and discuss why the potential generalisations seem so much more involved than what we have done here.

Setting. We consider the theory of the standard reference on homotopy type theory, that is, the textbook [14]. To summarise, we need a version of intensional Martin-Löf type theory with Σ, Π, and identity types. In addition, we assume that the theory has a univalent universe, and that there are truncation operators $\|-\|_n$ for all $n \geq -1$, with the canonical projections $|-| : A \to \|A\|_n$. This concept is explained in detail in [14, Chap. 7.3]). The statement and the first proof that we give do not need higher inductive types [14, Chap. 6] other than the truncations, while the second proof that we give makes heavy use of such a higher inductive type.

Agda Formalisation. We have formalised the main result, together with the "elementary" proof (Section 3) and the sample application (Section 5), in Agda [6]. The source code can be found on GitHub, at `github.com/pcapriotti/agda-base/tree/trunc`. The results of this paper are contained in the module `hott.truncation.elim`. A browsable HTML version of the formalisation can be accessed at `paolocapriotti.com/agda-base/trunc/hott/truncation/elim.html`. We encourage a reader who is not familiar with Agda to have a look at the latter, which does not need any software apart from a web browser. For all the technical details, we refer to the readme file in the repository.

On a minor note, we have chosen not to make use of the common (but, as far as we know, not justified by a formal argument) hack that makes truncations satisfy the judgmental computation rule. As we wanted our formalisation to be readable, this has required us to think of some implementation strategies that make the code in this setting more elegant than the "straightforward" formalisation approaches.

2 The Statement of the Theorem

Let us begin by clarifying some notation. In general, we stick closely to the terminology of the standard reference on the topic, the textbook [14]. We write $\Pi_{a:A} B(a)$ for Π-types as it is done there, but $\Sigma\,(a : A)\,.\,B(a)$ for Σ-types.[2] For better readability, we uncurry implicitly and write $f(a, b) : C$, even if f is a function of type $A \to B \to C$. Instead of $\lambda h.h \circ g$, we write $\_ \circ g$. By the *distributivity law of* Σ *and* Π, we mean the well-known equivalence

$$\Pi_{a:A}\Sigma\,(b : B(a))\,.\,C(a, b) \;\simeq\; \Sigma\,(g : \Pi_{a:A} B(a))\,.\,\Pi_{a:A} C(a, g(a)), \tag{1}$$

sometimes called the *type-theoretic axiom of choice*. As it is standard [14], we write $\mathsf{is}\text{-}n\text{-}\mathsf{type}(A)$ for the propositional type expressing that A is n-truncated if $n \geq -2$ is an integer, defined by

$$\mathsf{is}\text{-}(-2)\text{-}\mathsf{type}(A) :\equiv \Sigma\,(a_0 : A)\,.\,\Pi_{a:A} a = a_0 \tag{2}$$

$$\mathsf{is}\text{-}(n+1)\text{-}\mathsf{type}(A) :\equiv \Pi_{a_1,a_2:A}\mathsf{is}\text{-}n\text{-}\mathsf{type}(a_1 = a_2), \tag{3}$$

and the special case when n is -2 ("A is contractible") is also written as $\mathsf{isContr}(A)$. We assume that there is a universe $\mathcal{U}$, and we write $\mathcal{U}^n$ for the type (or "universe") of n-types in $\mathcal{U}$ (cf. [14, Chap. 7.1]),

$$\mathcal{U}^n :\equiv \Sigma\,(X : \mathcal{U})\,.\,\mathsf{is}\text{-}n\text{-}\mathsf{type}(X). \tag{4}$$

[2] This seemingly inconsistent notation is intentional: we sometimes have nested Σ-types, e.g. $\Sigma\,(a : A)\,.\,\Sigma\,(b : B(a))\,.\,C(a, b)$, and we view the components as "equally valued"; thus, writing exactly one component bigger than the others would not look correct.

Further, we write $\mathcal{U}_\bullet$ for the type (or "universe") of pointed types in $\mathcal{U}$ (cf. [14, Def. 2.1.7]),

$$\mathcal{U}_\bullet :\equiv \Sigma\,(X : \mathcal{U})\,.\,X. \tag{5}$$

If we have a type A and a pointed type (B, b), together with a function $f : A \to B$, we say that "f is null" if it is constantly b, that is,

$$\mathsf{isNull}(f) :\equiv \Pi_{x:A} b = f(x). \tag{6}$$

Recall that there is an endofunction on $\mathcal{U}_\bullet$, the *loop space function* Ω,

$$\Omega(A, a) :\equiv (a = a, \mathsf{refl}_a)\,. \tag{7}$$

For any natural number n, we can iterate this endofunction n times, for which we write Ω^n. Instead of $\pi_1\,(\Omega^n(A, a))$ and $\pi_1((\Omega(A, a)))$, we simply write $\Omega^n_t(A, a)$ and $\Omega_t(A, a)$ if we want to talk about the underlying type (i.e. ignore the point). Further, given two types A and B together with any function $f : A \to B$ and a point $a : A$, we have a function

$$\mathsf{ap}_{f,a} : \Omega_t(A, a) \to \Omega_t(B, f(a)). \tag{8}$$

In the same way, we have (given A, B, f as before) $\mathsf{ap}^n_{f,a} : \Omega^n_t(A, a) \to \Omega^n_t(B, f(a))$, and Ω is really an endofunctor in some appropriate sense.[3]

Our result can now be stated as follows:

▶ **Theorem 1.** *Let $n \geq -1$ be a number, A a type, and B an $(n + 1)$-type. Assume that $f : A \to B$ is a function. Then, f can be factored through the n-truncation, that is*

$$\Sigma\,(f' : \|A\|_n \to B)\,.\,f' \circ |-| = f, \tag{9}$$

if and only if $\mathsf{ap}^{n+1}_{f,a}$ is null for every a,

$$\Pi_{a:A}\mathsf{isNull}(\mathsf{ap}^{n+1}_{f,a}), \tag{10}$$

and both of the types (9) *and* (10) *are propositional.*

An immediate corollary tells us how we can eliminate out of truncations:

▶ **Corollary 2.** *Assume we have n, A and B as in Theorem 1. If we want to construct a function $\|A\|_n \to B$, it suffices to find a function $f : A \to B$ which satisfies $\Pi_{a:A}\mathsf{isNull}(\mathsf{ap}^{n+1}_{f,a})$.*

Before approaching a proof of Theorem 1, let us have a look at two special cases, namely the cases $n \equiv -1$ and $n \equiv 0$. The first case is known [10] and will serve as the base case for the two general proofs presented later. The second case is not strictly necessary, but serves to exemplify the techniques used in the "HIT proof" (Section 4).

The case $\mathbf{n \equiv -1}$. The simplified statement of Theorem 1 reads in this case as follows: Assume we are given a type A and a 0-type B (often called a *set*). A function $f : A \to B$ factors through the propositional truncation if and only if

$$\Pi_{x,y:A} f(x) = f(y). \tag{11}$$

This follows easily from previous work, e.g. [8, Prop. 2.2]. It is a pleasant surprise that "$\mathsf{ap}^0_{f,a}$ is null for all a", simply by unfolding our definitions, simplifies to (11), which is "f is constant" in the sense of [10].[4]

[3] Of course, $\mathsf{ap}_{f,a}$ is its action on the morphism f and could thus rightfully be called $\Omega(f, a)$.

[4] In the simplified formulation, we have omitted the part that the two logically equivalent types are propositional. This is easy to see here, and will in the general case be part of the proof.

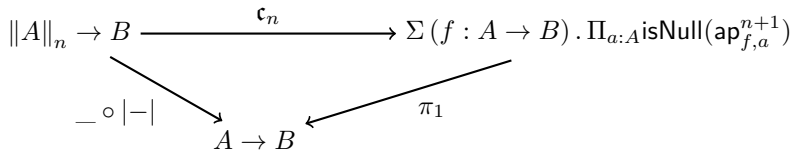


Figure 1 The canonical map $\mathfrak{c}_n$ as map between fibres.

The case $\mathbf{n \equiv 0}$. Here, our result (Theorem 1) implies that, for any type A and 1-type B, a function $f : A \to B$ factors through $\|A\|_0$ if and only if, for all $a : A$ and $p : a = a$, we have that $\mathsf{ap}_{f,a}(p)$ equals $\mathsf{refl}_{f(a)}$. As Shulman has remarked in an online discussion (in the comment section of a blog post [5]), this follows from the *Rezk completion* [1]: Let $\tilde{A}$ be the precategory with the type A of objects and $\mathsf{hom}(a_1, a_2) :\equiv \|a_1 =_A a_2\|_{-1}$, and let $\tilde{B}$ be the category with B as objects and $\mathsf{hom}(b_1, b_2) :\equiv (b_1 =_B b_2)$. Then, f with the condition $\Pi_{a:A}\mathsf{isNull}(\mathsf{ap}_{f,a})$ gives (already using the case $n \equiv -1$) rise to a functor $\tilde{A} \to \tilde{B}$. Such a functor generates a functor between the Rezk completion of $\tilde{A}$ and the category $\tilde{B}$, and the former happens to be $\|A\|_0$.

In the remainder of the current section, we give a simple technical construction which essentially serves as a reformulation of Theorem 1 and which is necessary for both the elementary and the HIT proof. For types A and B, assume we are given a function $g : \|A\|_n \to B$. We can consider the composition $A \xrightarrow{|-|} \|A\|_n \xrightarrow{g} B$. For any $a : A$ we have, by functoriality of Ω^{n+1}, that the composition

$$\Omega^{n+1}_t(A, a) \xrightarrow{\mathsf{ap}^{n+1}_{|-|,a}} \Omega^{n+1}_t(\|A\|_n, |a|) \xrightarrow{\mathsf{ap}^{n+1}_{g,|a|}} \Omega^{n+1}_t(B, g(|a|)) \tag{12}$$

is equal to $\mathsf{ap}^{n+1}_{g\circ|-|,a}$. But $\Omega^{n+1}_t(\|A\|_n, |a|)$ is contractible ([14, Thm. 7.2.9]), and $\mathsf{ap}^{n+1}_{g,|a|}$ clearly maps its unique element to the basepoint of $\Omega^{n+1}(B, g(|a|))$. Therefore, $\mathsf{ap}^{n+1}_{g\circ|-|,a}$ is null. From this construction, we get a canonical function

$$\mathfrak{c}_n : (\|A\|_n \to B) \to \Sigma\,(f : A \to B)\,.\,\left(\Pi_{a:A}\mathsf{isNull}(\mathsf{ap}^{n+1}_{f,a})\right). \tag{13}$$

We then claim the following:

▶ **Lemma 3** ("Total space" formulation of Theorem 1)**.** *For any $n \geq -1$, any type A and any $(n+1)$-type B, the types $\|A\|_n \to B$ and $\Sigma\,(f : A \to B)\,.\,\Pi_{a:A}\mathsf{isNull}(\mathsf{ap}^{n+1}_{f,a})$ are equivalent, and the equivalence is given by the canonical function $\mathfrak{c}_n$.*

It is easy to see that Lemma 3 does indeed imply, and is nearly immediately equivalent to, Theorem 1. Consider the triangle shown in Figure 1, where the top horizontal map is the canonical map $\mathfrak{c}_n$, the left one is composition with $|-|$, and the right one is simply the projection. The triangle clearly commutes (judgmentally) by construction. Let us fix some function $f : A \to B$. The fibre (or "inverse image") over f is, in the case of $\_ \circ |-|$, exactly (9), i.e. the statement that f can be lifted. In the second case, the fibre is (10). Therefore, $\mathfrak{c}_n$ induces an equivalence of the two fibres, which implies that $\mathfrak{c}_n$ itself is an equivalence (see [14, Thm. 4.7.7]).

3 The "Elementary" Proof

In this section, we give our first proof of Lemma 3 (and thereby of Theorem 1). This does not need higher inductive types apart from truncations that already appear in the statement.

The idea is to not prove the result for *any* type A first, but only for an n-connected one.[5] Afterwards, we generalise this to arbitrary types, by splitting the type into its "connected components" and gluing together the constructions for the components.

▶ **Lemma 4.** *If $n \geq -1$ be a number, A an n-connected type, and B be an $(n+1)$-type, the canonical map $\mathfrak{c}_n$ is an equivalence.*

Proof. We do induction on n. As already discussed above, the case that n is -1 is known (e.g. [8, Prop. 2.2]).

Let now $n \geq 0$ be any given number. Note that, due to the assumption that $\|A\|_n$ is contractible, we have a unique element $x_0 : \|A\|_n$, the type $\|A\|_n \to B$ is actually equivalent to B, and any function $g : \|A\|_n \to B$ is uniquely specified by its value $g(x_0)$.

The claim of the lemma is propositional. Applying the eliminator of $\|A\|_n$, we may not only assume that we are given $x_0 : \|A\|_n$, but we can also assume a point $a : A$. A potential inverse of $\mathfrak{c}_n$ is then given by[6]

$$\mathfrak{d}_n : \left(\Sigma\,(f : A \to B)\,.\,\Pi_{a:A}\mathsf{isNull}(\mathsf{ap}^{n+1}_{f,a})\right) \to (\|A\|_n \to B) \tag{14}$$

$$\mathfrak{d}_n(f,p) :\equiv \lambda\_.f(a). \tag{15}$$

To show that $\mathfrak{c}_n$ and $\mathfrak{d}_n$ are inverses, we check that both compositions are the identities. One direction is easy: for any $g : \|A\|_n \to B$, we have

$$\mathfrak{d}_n(\mathfrak{c}_n(g))(x_0) \equiv g(|a|), \tag{16}$$

and the latter is equal to $g(x_0)$.

For the other direction, assume we have $f : A \to B$ together with a proof q. We need to show $(f,q) = \mathfrak{c}_n(\mathfrak{d}_n(f,q))$. Fortunately, the equality of the two second components is automatic thanks to the fact that $\mathsf{isNull}(\mathsf{ap}^{n+1}_{f,a})$ is propositional, and we only need to prove the equality of f and $\pi_1(\mathfrak{c}_n(\mathfrak{d}_n(f,q)))$. We observe that the latter expression computes to $\lambda\_.f(a)$. Thus, our goal is to show that, for any $a' : A$, we have $f(a) = f(a')$.

We use the induction hypothesis with $(a = a')$ for A, and $f(a) = f(a')$ for B. By the connectedness assumption on A, the type $|a| = |a'|$ is contractible. Consequently, the type $\|a = a'\|_{n-1}$ is contractible ([14, Thm. 7.3.12], note that this theorem depends on the univalence axiom). Put differently, $(a = a')$ is $(n-1)$-connected. As B is an $(n+1)$-type, we know that $f(a) = f(a')$ is n-truncated. By the induction hypothesis, it is hence enough to construct an element of

$$\Sigma\,(k : a = a' \to f(a) = f(a'))\,.\,\Pi_{p:a=a'}\mathsf{isNull}(\mathsf{ap}^n_{k,p}). \tag{17}$$

For k, we choose ap_f. By path induction, we may assume that p is refl_a. Thus, we need to show that $\mathsf{ap}^n_{\mathsf{ap}_{f,a},\mathsf{refl}_a}$ is null. This term is equal to $\mathsf{ap}^{n+1}_{f,a}$.[7] The condition that this function null is exactly what is given by $q(a')$. ◀

To move from n-connected to arbitrary types A, we simply split a type into n-connected components. This is very intuitive for $n \equiv 0$, in which case we use that any type (or "space") can be viewed as the "disjoint sum" of its connected components. To be precise, an element

[5] Recall that a type A is n-connected if $\|A\|_n$ is contractible [14, Def. 7.5.1].
[6] We use _ if we do not need to give the bound variable a name.
[7] Depending on the the exact definition of ap^n, this can hold judgmentally, but can also be rather involved. We refer to our formalisation for technical details.

of a component is a point of A together with a proof that it is in the component. For $n \equiv 0$, this proof is propositional. For higher n, it is not. This makes the general case less intuitive and hard to picture. In fact, the proof determines in which component the element is, which makes it seem circular. Fortunately, it is easier to write down the type-theoretic argument than picturing the topological intuition, as we will see in the following lemma.

▶ **Lemma 5.** *For any type A and number n, we define the family of n-connected components,*

$$\mathsf{conn}_n : \|A\|_n \to \mathcal{U} \tag{18}$$

$$\mathsf{conn}_n(x) :\equiv \Sigma\,(a : A)\,.\,x =_{\|A\|_n} |a|. \tag{19}$$

Then, for any $x : \|A\|_n$, the type $\mathsf{conn}_n(x)$ is n-connected. Further, "choosing an n-connected component and then a point in this component" corresponds to "choosing a point", that is,

$$\Sigma\,(x : \|A\|_n)\,.\,\mathsf{conn}_n(x) \;\simeq\; A. \tag{20}$$

Proof. This is easy and standard. For the first part, we claim that the equivalence

$$\left\|\Sigma\,(a : A)\,.\,x =_{\|A\|_n} |a|\right\|_n \;\simeq\; \Sigma\,(y : \|A\|_n)\,.\,x =_{\|A\|_n} y \tag{21}$$

holds, where the left-hand type is $\|\mathsf{conn}_n(x)\|_n$ by definition, and the right-hand type has the form of a *singleton*.[8] For both directions of (21), we apply the dependent eliminator of $\|-\|_n$. From left to right, we map $|(a, p)|$ to $(|a|, p)$. From right to left, we map $(|a|, p)$ to $(|a, p|)$. For an alternative proof, see [14, Cor. 7.5.8].

To see that the equivalence (20) holds, it is enough to unfold the definition of conn_n, and use that in $\Sigma\,(x : \|A\|_n)\,.\,\Sigma\,(a : A)\,.\,x =_{\|A\|_n} |a|$, the first and the third component form a singleton. ◀

Finally, we can complete the first proof of our main result:

"Elementary" proof of Lemma 3. Assume we have n, A, and B as in the statement. The preceding two lemmata tell us that, for any $x : \|A\|_n$, the canonical map

$$\mathfrak{c}_n^x : B \to \left(\Sigma\,(f_x : \mathsf{conn}_n(x) \to B)\,.\,\Pi_{y:\mathsf{conn}_n(x)}\mathsf{isNull}(\mathsf{ap}^{n+1}_{f_x,y})\right) \tag{22}$$

is an equivalence (note that we have omitted the contractible type $\|\mathsf{conn}_n(x)\|_n$ in the domain of $\mathfrak{c}_n^x$). A family of equivalences gives rise to an equivalence of families, so that we get that the map

$$\tilde{\mathfrak{c}}_n : (\|A\|_n \to B) \;\to\; \left(\Pi_{x:\|A\|_n}\Sigma\,(g_x : \mathsf{conn}_n(x) \to B)\,.\,\Pi_{y:\mathsf{conn}_n(x)}\mathsf{isNull}(\mathsf{ap}^{n+1}_{g,y})\right) \tag{23}$$

$$\tilde{\mathfrak{c}}_n(k) :\equiv \lambda x.\mathfrak{c}_n^x(k(x)) \tag{24}$$

is also an equivalence.

All we need at this point is an equivalence from the codomain of the function (24) to the type stated in the theorem, i.e. $\Sigma\,(f : A \to B)\,.\,\Pi_{a:A}\mathsf{isNull}(\mathsf{ap}^{n+1}_{f,a})$, and the composition of (24) and this equivalence has to be the canonical map $\mathfrak{c}_n$. We calculate:

$$\Pi_{x:\|A\|_n}\Sigma\,(g_x : \mathsf{conn}_n(x) \to B)\,.\,\Pi_{y:\mathsf{conn}_n(x)}\mathsf{isNull}(\mathsf{ap}^{n+1}_{g_x,y}) \tag{25}$$

[8] If $z_0 : Z$ is some point of some type, we call any type of the form $\Sigma\,(z : Z)\,.\,z = z_0$ a *singleton*. It is well-known that singletons are contractible and therefore "neutral" components of Σ-types, which we use here and later.

(by the distributivity law)

$$\simeq \quad \Sigma\left(g : \Pi_{x:\|A\|_n}(\mathsf{conn}_n(x) \to B)\right).\, \Pi_{x:\|A\|_n} \Pi_{y:\mathsf{conn}_n(x)} \mathsf{isNull}(\mathsf{ap}^{n+1}_{g(x),y}) \tag{26}$$

(by currying and using the canonical equivalence (20))

$$\simeq \quad \Sigma\,(h : A \to B).\, \Pi_{a:A} \mathsf{isNull}(\mathsf{ap}^{n+1}_{\lambda y:\mathsf{conn}_n(|a|).h(\pi_1 y),(a,\mathsf{refl}_{|a|})}) \tag{27}$$

Fortunately, the (pointed) types $\Omega^{n+1}(\mathsf{conn}_n(|a|), (a, \mathsf{refl}_{|a|}))$ and $\Omega^{n+1}(A, a)$ are equivalent, with the equivalence being $\mathsf{ap}^{n+1}_{\pi_1}$; this is an easy technical statement that follows from [11, Lem. 5.1]. If we compose $\mathsf{ap}^{n+1}_{\lambda y:\mathsf{conn}_n(|a|).h(\pi_1 y),(a,\mathsf{refl}_{|a|})}$ with the inverse of this equivalence, functoriality of ap^{n+1} allows us to simplify the expression.

$$\simeq \quad \Sigma\,(h : A \to B).\, \Pi_{a:A} \mathsf{isNull}(\mathsf{ap}^{n+1}_{h,a}) \tag{28}$$

We need to check that the composition of $\tilde{\mathfrak{c}}_n$ with this equivalence is indeed the canonical function $\mathfrak{c}_n$. This is immediate as we only need to check that the first component (the map $A \to B$) turns out to be the correct function, as the second component is propositional. ◀

4 The "HIT Proof"

Our second proof is fairly technical. We construct a higher inductive type with a suitable elimination property and show that it is equivalent to $\|A\|_n$. As a preparation, we show a small lemma. It is a part of a theorem that has been introduced in [9], where it is described as *local generalised Hedberg argument*.

▶ **Lemma 6** (main part of [9, Thm. 3.2.1])**.** *Let (A, a_0) be a pointed type. Assume further that P is a* pointed *family of $(n-1)$-types over (A, a_0), that is, a family $P : A \to \mathcal{U}^{n-1}$ with a point $p_0 : P(a_0)$. If $P(a)$ implies that a_0 is equal to a, i.e. $m : \Pi_{a:A} P(a) \to a_0 = a$, then A is "locally an n-type" in the sense that $\Omega^{n+1}(A, a_0)$ is contractible.*[9]

Proof sketch. Consider the following composition of three maps, for any $a : A$:

$$a_0 = a \xrightarrow{q \mapsto \mathsf{transport}^P(q, p_0)} P(a) \xrightarrow{m_a} a_0 = a \xrightarrow{q \mapsto m_{a_0}(p_0) \,\cdot\, q} a_0 = a$$

By path induction, we easily see that these maps make $a_0 = a$ a retract of $P(a)$. Hence, the former is $(n-1)$-truncated [14, Thm. 7.1.4], which shows the claim [14, Thm. 7.2.9]. ◀

We are ready to define the higher inductive type that plays the central role in the second proof of Lemma 3. For the following definition and for the rest of the section, we fix a type A and a number $n \geq -1$.

▶ **Definition 7.** Define the higher inductive type H, which depends on A and n, as given by the constructors

$$\eta : A \to H \tag{29}$$

$$\epsilon : \Pi_{a,b:A}\left(\|a = b\|_{n-1} \to \eta(a) = \eta(b)\right) \tag{30}$$

$$\delta : \Pi_{a:A}\left(\mathsf{refl}_{\eta(a)} =_{\eta(a)=\eta(a)} \epsilon(a, a, |\mathsf{refl}_a|)\right) \tag{31}$$

$$t : \mathsf{is}\text{-}(n+1)\text{-}\mathsf{type}(H). \tag{32}$$

[9] This "local" form directly implies the "global" form: We can consider a relation $R : A \times A \to \mathcal{U}^{n-1}$ which implies identity and which has points $r_a : R(a, a)$ for all $a : A$; then, the lemma shows that A is an n-type.

The complicated looking constructors ϵ and δ are more intuitive than they looks at first sight. If we have $(a = b)$, we of course always get a proof of $\eta(a) = \eta(b)$ using ap_η. The constructor ϵ says that $\|a = b\|_{n-1}$ is sufficient, while δ ensures that ϵ is really a lifting of ap_η through $\|a = b\|_{n-1}$. This is because we could have used the expanded form

$$\delta' : \Pi_{a,b:A}\Pi_{p:a=b}\left(\mathsf{ap}_\eta(p) =_{\eta(a)=\eta(b)} \epsilon(a, b, |p|)\right), \tag{33}$$

instead of the constructor δ. By path induction on p, the type (33) is easily seen to be equivalent to the original type (31). While (33) might look more regular next to (30), we choose (31) just for simplicity.

The recursion principle for H is straightforward to write down. Given some $(n+1)$-type B, we need a function $f : A \to B$, together with a function $k : \Pi_{a,b:A}(\|a = b\|_{n-1}) \to f(a) = f(b)$ and a proof $h : \Pi_{a:A}\mathsf{refl}_{f(a)} =_{f(a)=f(a)} k(a, a, |\mathsf{refl}_{f(a)}|)$, we get a function $H \to B$ with the expected properties. It is more involved, nevertheless not inherently difficult, to state the induction principle following the standard ("intuitive") approach as used in [14, Chap. 6]. Given an $(n + 1)$-truncated family $P : H \to \mathcal{U}^{n+1}$, in order to prove $\Pi_{x:H}P(x)$, we need

$$\overline{\eta} : \Pi_{a:A}P(\eta(a)) \tag{34}$$

$$\overline{\epsilon} : \Pi_{a,b:A}\Pi_{q:\|a=b\|_{n-1}}\mathsf{transport}^P\left(\epsilon(a, b, q), \overline{\eta}(a)\right) =_{P(\eta(b))} \overline{\eta}(b) \tag{35}$$

$$\overline{\delta} : \Pi_{a:A}\left(\mathsf{transport}^{\lambda r.\mathsf{transport}^P(r,\overline{\eta}(a))=\overline{\eta}(a)}\left(\delta(a), \mathsf{refl}_{\overline{\eta}(a)}\right) = \overline{\epsilon}(a, a, |\mathsf{refl}_a|)\right). \tag{36}$$

The above type expressions look rather involved. Fortunately, we do not need to deal too much with them at all because we are only interested in the case that P is n-truncated (instead of, more generally, $(n + 1)$-truncated), which enables us to use the following observation:

▶ **Lemma 8** (Restricted dep. universal property of H). *Given A and $n \geq -1$ as above and a family of n-types, $P : H \to \mathcal{U}^n$, the canonical map*

$$\Pi_{x:H}P(x) \xrightarrow{-\circ\eta} \Pi_{a:A}P(\eta(a)) \tag{37}$$

is an equivalence.

Proof. As P is a family of n-types, the type $\mathsf{transport}^P\left(\epsilon(a, b, q), \overline{\eta}(a)\right) =_{P(\eta(b))} \overline{\eta}(b)$, appearing in (35) as the target of $\overline{\epsilon}$, is $(n-1)$-truncated. By the standard universal property of the $(n-1)$-truncation, we may thus assume that the q in the type (35) is of the form $|p|$ with $p : a = b$, and then do path induction on p. This shows that the type of $\overline{\epsilon}$ is equivalent to

$$\overline{\epsilon}'' : \Pi_{a:A}\mathsf{transport}^P\left(\epsilon(a, a, |\mathsf{refl}_a|), \overline{\eta}(a)\right) =_{P(\eta(a))} \overline{\eta}(a). \tag{38}$$

Under this equivalence, the type of $\overline{\delta}$ becomes

$$\overline{\delta}'' : \Pi_{a:A}\left(\mathsf{transport}^{\lambda r.\mathsf{transport}^P(r,\overline{\eta}(a))=\overline{\eta}(a)}\left(\delta(a), \mathsf{refl}_{\overline{\eta}(a)}\right) = \overline{\epsilon}''(a)\right). \tag{39}$$

We see that the dependent pair of (38) and (39) forms a family of singletons. Therefore, there is always a canonical and unique choice for $\overline{\epsilon}$ and $\overline{\delta}$. The induction principle can therefore be simplified to only (34). Let us write $\mathsf{rind} : \Pi_{a:A}P(\eta(a)) \to \Pi_{x:H}P(x)$ for this *restricted induction principle*. It is easy to check that rind is indeed an inverse of the map $\_ \circ \eta$:

- For any $f : \Pi_{a:A}P(\eta(a))$ and $a : A$, the expression $(\mathsf{rind}(f) \circ \eta)(a)$ can be reduced to $f(a)$.
- For any $g : \Pi_{x:H}P(x)$, assume $x : H$. We need to show $(\mathsf{rind}(g \circ \eta))(x) = g(x)$. Using the restricted induction principle, we may assume $x \equiv \eta(a)$, and the left side can be reduced to the right side of the equation. ◀

This allows us to conclude the following crucial property of H:

▶ **Lemma 9.** *The type H is n-truncated.*

Proof. It suffices to show that $\Omega^{n+1}(H, x)$ is contractible for all $x : H$ [14, Lem. 7.2.9]. The restricted induction principle of H tells us that, in order to show $P(x) :\equiv \mathsf{isContr}\left(\Omega^{n+1}(H, x)\right)$ for all x, we only need to prove $P(\eta(a_0))$ for any $a_0 : A$. Let us define a type family $Q : H \to \mathcal{U}^{n-1}$ using the restricted induction principle, $Q(\eta(a)) :\equiv \|a_0 = a\|_{n-1}$. This family is trivially inhabited at a_0. We want to show that Q implies local equality in the sense of $\Pi_{x:H}\left(Q(x) \to \eta(a_0) = x\right)$, and as this type family is n-truncated, we apply the restricted induction principle again and the goal becomes

$$\Pi_{a:A}\left(Q(\eta(a)) \to \eta(a_0) = \eta(a)\right). \tag{40}$$

By definition of Q, this is exactly given by the constructor ϵ, applied on a_0 and a.

This allows us to conclude, by Lemma 6, that H is n-truncated, as claimed. ◀

It is straightforward and standard that an n-truncated type which satisfies the dependent eliminating principle of $\|A\|_n$ is necessarily equivalent to $\|A\|_n$, and we record:

▶ **Corollary 10.** *The types H and $\|A\|_n$ are equivalent.*

At the same time, we have the following:

▶ **Lemma 11** (Universal property of H)**.** *For any $(n+1)$-type B, the type of functions $H \to B$ is equivalent to*

$$\Sigma\left(f : A \to B\right).\Sigma\left(e : \Pi_{a,b:A}\|a = b\|_{n-1} \to f(a) = f(b)\right).\left(d : \Pi_{a:A}\mathsf{refl}_{f(a)} = e(a, a, |\mathsf{refl}_a|)\right). \tag{41}$$

Proof sketch. The proof of deriving this form of universal property from the induction principle is standard. The map from $H \to B$ into the stated type is more or less composition with the constructors; for any $k : H \to B$, we get

$$(f, e, d) :\equiv \left(k \circ \eta\,,\ \mathsf{ap}_k \circ \epsilon\,,\ \lambda a.\mathsf{ap}_{\mathsf{ap}_k}(\delta(a))\right). \tag{42}$$

The map in the other direction is exactly the recursion principle of H. That they are mutually inverse corresponds to the computation (β) rule respectively the uniqueness (η) rule of H. ◀

Finally, we can complete the second proof of our main result:

"HIT proof" of Lemma 3. We do induction on n. The base case ($n \equiv -1$) is, as before, just what we have discussed in Section 2. For higher n, we have the following chain of equivalences:

$$\|A\|_n \to B \tag{43}$$

(by Corollary 10)

$$\simeq \quad H \to B \tag{44}$$

(by Lemma 11)

$$\simeq \quad \Sigma\left(f : A \to B\right).\Sigma\left(e : \Pi_{a,b:A}\|a = b\|_{n-1} \to f(a) = f(b)\right). \left(\Pi_{a:A}\mathsf{refl}_{f(a)} = e(a, a, |\mathsf{refl}_a|)\right) \tag{45}$$

(by "inverse path induction")

$$\simeq \quad \Sigma\,(f : A \to B)\,.\,\Sigma\left(e : \Pi_{a,b:A}\|a = b\|_{n-1} \to f(a) = f(b)\right).\\ \left(\Pi_{a,b:A}\Pi_{p:a=b}\mathsf{ap}_f p = e(a,b,|p|)\right) \tag{46}$$

(by the distributivity law)

$$\simeq \quad \Sigma\,(f : A \to B)\,.\,\Pi_{a,b:A}\big(\Sigma\left(e' : \|a = b\|_{n-1} \to f(a) = f(b)\right).\\ \Pi_{p:a=b}\mathsf{ap}_f p = e'(|p|)\big) \tag{47}$$

Now we exchange e' by $(e_1, e_2) :\equiv \mathfrak{c}_{n-1}(e')$ using the induction hypothesis, and thus we need to apply $\mathfrak{c}_{n-1}^{-1}$ to that term in the last component. Fortunately, it follows from the definition of $\mathfrak{c}_{n-1}$ that $\_ \circ \mathfrak{c}_{n-1} \equiv \pi_1 \circ |-|$, hence we can replace $e'(|p|)$ with simply $e_1(p)$:

$$\simeq \quad \Sigma\,(f : A \to B)\,.\,\Pi_{a,b:A}\big(\Sigma\,(e_1 : a = b \to f(a) = f(b))\,.\,\Sigma\left(e_2 : \Pi_{p:a=b}\mathsf{isNull}(\mathsf{ap}^n_{e_1,p})\right).\\ \left(\Pi_{p:a=b}\mathsf{ap}_f p = e_1(p)\right)\big) \tag{48}$$

The term e_1 and the very last (unnamed) component form a singleton and can be removed:

$$\simeq \quad \Sigma\,(f : A \to B)\,.\,\left(\Pi_{a,b:A}\Pi_{p:a=b}\mathsf{isNull}(\mathsf{ap}^n_{\mathsf{ap}_f,p})\right) \tag{49}$$

(by "path induction")

$$\simeq \quad \Sigma\,(f : A \to B)\,.\,\left(\Pi_{a:A}\mathsf{isNull}(\mathsf{ap}^n_{\mathsf{ap}_f,\mathsf{refl}_{f(a)}})\right) \tag{50}$$

(as $\mathsf{ap}^n_{\mathsf{ap}_f,\mathsf{refl}_a}$ is the same as $\mathsf{ap}^{n+1}_{f,a}$ – the footnote on page 365 applies)

$$\simeq \quad \Sigma\,(f : A \to B)\,.\,\left(\Pi_{a:A}\mathsf{isNull}(\mathsf{ap}^{n+1}_{f,\mathsf{refl}_a})\right). \tag{51}$$

Finally, we need to check that the constructed equivalence is indeed the canonical function $\mathfrak{c}_n$. Fortunately, the second (and more involved) part $\Pi_{a:A}\mathsf{isNull}(\mathsf{ap}^{n+1}_{f,\mathsf{refl}_a})$ is propositional. It is therefore enough to check that any map $g : \|A\|_n \to B$ gets, by the constructed equivalence, mapped to a pair in (51) of which *the first component* is $g \circ |-|$. But the first component is constructed in the very first step, where Lemma 11 is applied, and, looking at the proof of Lemma 11, it is indeed simply composition with $|-|$. ◀

5 A Sample Application: Set-Based Groupoids

A set-theoretic ω-groupoid has, in the "globular" formulation, ω-many levels: At level 0, it has a collection of objects (or 0-cells); for any two objects, it has a collection of 1-morphisms (1-cells); for any two 1-morphisms, there is a collection of 2-morphisms (2-cells), and so on. As recalled in the introduction, types indeed are such ω-groupoids meta-theoretically. It is intuitive to ask how much of this can be internalised. Defining a weak ω-groupoid in type theory is already very hard [2, 3]: one would want a 0-type (i.e. a *set*) A_0 of 0-cells, a *set* A_1 of 1-cells which is indexed twice over A_0, and so on. Even if one has such a definition at hand, it is implausible to expect that one can define the "fundamental ω-groupoid" of a type. As Altenkirch, Li and Rypacek [2] mention, they are unable to construct such an ω-groupoid, which in their terminology is called $\mathsf{Id}\omega$. The Ph.D. thesis of the second-named author of the current paper includes a precise negative statement [9, Sec. 9.4.1] which shows that a construction in the sense of [2] is impossible in all non-trivial cases. The argument given

there indicates that a fundamental reason why we cannot even define A_1 is that we want A_1 to be indexed twice over A_0.

However, we know that the whole higher structure of types is in some sense determined by the loop spaces, as opposed to the path spaces. It seems therefore reasonable to consider a more modest variation where we index A_1 only once over A_0, with the intention that $A_1(a_0)$ represents the loop space over a_0. This has the further advantage that we can assume that A_0 is $\|A\|_0$; with double-indexed A_1, it would be possible that elements $a, b : A_0$ are not equal in A_0, but "made equal" by an element of $A_1(a, b)$. As a further simplification, we only consider the question whether a type can be represented in two levels, i.e. with $A_0 \equiv \|A\|_0$ and A_1.

▶ **Definition 12.** We call a type A *set-based representable* if the function

$$\omega_A : A \to \mathcal{U} \tag{52}$$
$$\omega_A(a) :\equiv (a = a) \tag{53}$$

factors through $\|A\|_0$, i.e. if there is a single-indexed family $A_1 : \|A\|_0 \to \mathcal{U}$ of types which, for all $a : A$, satisfies $A_1(|a|) \simeq (a =_A a)$.

We also define the following simple notion:

▶ **Definition 13.** We say that a type A has *loop spaces with braidings* if, for all $a : A$ and $p, q : a = a$, we have $p \cdot q = q \cdot p$.

Examples of types which have loop spaces with braidings are sets (for which the condition is trivial), and, more interestingly, loop spaces themselves.

▶ **Theorem 14.** *Every* 1*-type whose loop spaces have braidings is set-based representable.*

Proof. As A is a 1-type, the function (52) takes sets as values; that is, in this case, we can assume that ω_A is of type $A \to \mathcal{U}^0$. Using that $\mathcal{U}^0$ is a 1-type [14, Thm. 7.1.11], we may apply Theorem 1 with $n \equiv 0$. We need to show that, for a fixed $a : A$, the function

$$\mathsf{ap}_{\omega_A, a} : \Omega_t(A, a) \to \Omega_t(\mathcal{U}, a = a) \tag{54}$$

is null. But $\mathsf{ap}_{\omega_A}(p)$ induces a *function* of type $(a = a) \to (a = a)$ (via the function that is called $\mathsf{idtoeqv}$ in [14], and projection), and by univalence, it is enough to show that this function does not depend on p. We claim that this function maps $q : a = a$ to $p^{-1} \cdot q \cdot p$. An easy way to prove this claim is considering the more general version of ap_{ω_A} that works on any path spaces (instead of loop spaces), and then doing path induction on p. Clearly, the braiding on $a = a$ is exactly what we need to justify that $p^{-1} \cdot q \cdot p$ does not depend on p. ◀

6 The Big Picture: Solved and Unsolved Cases

The "ordinary" universal property of the n-truncation can be recovered easily from Theorem 1. If, under the conditions of the statement, B is not only $(n+1)$-, but even n-truncated, the type $\Pi_{a:A}\mathsf{isNull}\left(\mathsf{ap}^{n+1}_{f,a}\right)$ becomes contractible, and the theorem says precisely that functions $A \to B$ are the same as functions $\|A\|_n \to B$, via composition with $|-|$. Theorem 1 is thus stronger than the "ordinary" universal property. However, we weaken the condition on B by only one single level, while [8] weakens it by arbitrary many levels, but only for the propositional truncation.

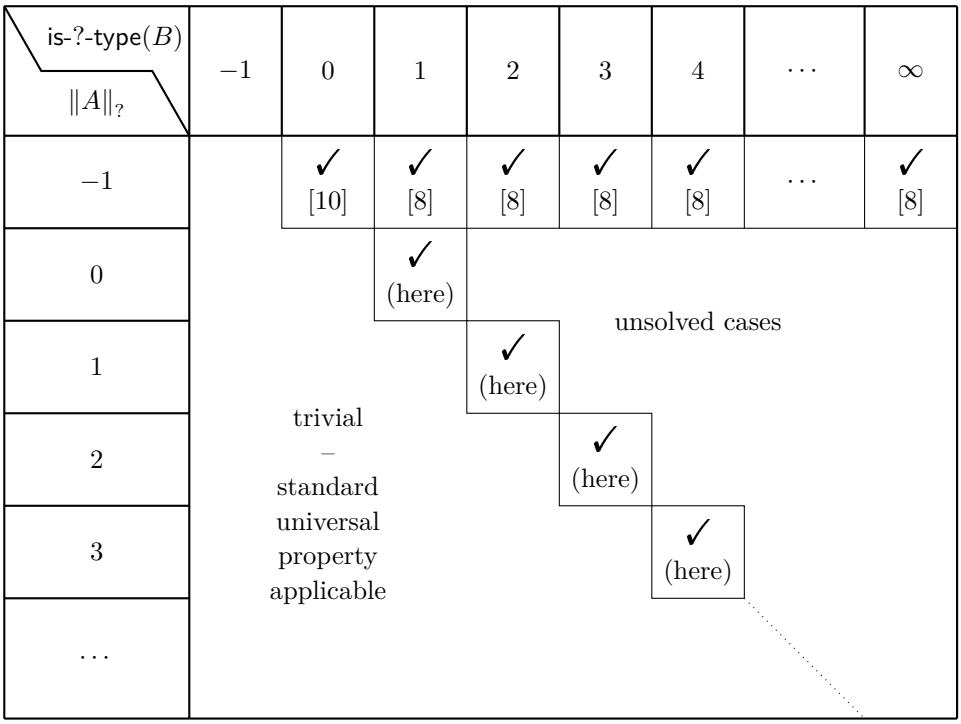


Figure 2 The universal property of $\|A\|_?$ with respect to ?-types: trivial, solved, and open cases.

Of course, the general question is: What is the universal property of $\|A\|_n$ with respect to m-types, i.e. how can we construct a map $\|A\|_n \to B$ for some m-type B? Put differently, given a function $f : A \to B$, how can we (by only imposing conditions on f, not on A or B) ensure that f factors through $\|A\|_n$? Figure 2 illustrates the current progress on this question. As indicated, the question is trivial if m is not greater than n. Two other families of cases are solved, those with $m \equiv n+1$ by the current paper, and $n \equiv -1$ by [8]. Note that the latter is not internalised in the way that the result of the current paper is, and it is not to be expected that an internalisation is possible in the considered type theory; and further, the case $n \equiv -1$, $m \equiv \infty$ (meaning that there is no condition at all on B) is solved, but only under the assumption of Reedy ω^{op}-limits.

The (probably) simplest case that is left open is the case $n \equiv 0$, $m \equiv 2$. So, let us consider a function $f : A \to B$, where B is 2-truncated. Which conditions do we have to impose on f to conclude that it factors through $\|A\|_0$? As is easy to show, if f factors through the 0-truncation, then ap_f factors through the (-1)-truncation. The necessary conditions for the latter have been worked out in [8], and we could thus try to impose them on ap_f (at all points). However, this does not work. In one aspect, the propositional truncation is a special case that is actually *harder* than the higher truncations, intuitively because loop spaces are always pointed[10] which we have already made use of in the definition of isNull. It turns out that in this "pointed" case one can get all these coherences (which make the result of [8] hard) for free. Instead, the higher groupoid structure of loop spaces induces a different sort

[10] This seems to correspond to the fact that the zeroth homotopy "group" is not a group, and does therefore not have a canonical element, which seems to occasionally make this special case harder in traditional topology as well.

of coherence problem. For example, it certainly *is* necessary that, for any $a : A$ and $p : a = a$, there is a proof $c_{a,p} : \mathsf{ap}_{f,a}(p) = \mathsf{refl}_{f(a)}$. From $c_{a,p}$, we can construct a proof that $\mathsf{ap}_{a,f}(p \cdot p)$ equals $\mathsf{refl}_{f(a)}$, using functoriality of $\mathsf{ap}_{f,a}$. If we want the family c to be "fully coherent", we have to force this proof to be the same as $c_{a,p \cdot p}$. The work [8] concludes with a precise conjecture of how *all* the required coherence conditions can be captured in the general case. At this time, it is unknown whether this can be used to fill in the missing parts of Figure 2.

Acknowledgements. We would like to thank Thorsten Altenkirch and Christian Sattler for fruitful discussions. The second-named author is grateful for the opportunity to discuss some of the ideas that have led to this paper with participants of several events, including the *Institut Henri Poincaré thematic trimester*. We further want to acknowledge that Michael Shulman has made the connection with the Rezk completion precise, and we thank the anonymous reviewers for their reports that have helped us improving this paper.

References

1 Benedikt Ahrens, Krzysztof Kapulkin, and Michael Shulman. Univalent categories and the Rezk completion. *Mathematical Structures in Computer Science (MSCS)*, pages 1–30, Jan 2015.

2 Thorsten Altenkirch, Nuo Li, and Ondrej Rypacek. Some constructions on ω-groupoids. In *Logical Frameworks and Meta-languages: Theory and Practice (LFMTP)*, 2014.

3 Thorsten Altenkirch and Ondrej Rypacek. A syntactical approach to weak ω-groupoids. In *Computer Science Logic (CSL)*, pages 16–30, 2012.

4 Steve Awodey and Andrej Bauer. Propositions as [types]. *Journal of Logic and Computation*, 14(4):447–471, 2004.

5 Paolo Capriotti. Higher lenses. Blog post at `homotopytypetheory.org`, 29 Apr 2014.

6 Paolo Capriotti, Nicolai Kraus, and Andrea Vezzosi. Functions out of higher truncations (Agda formalisation), Apr 2015. Available at `https://github.com/pcapriotti/agda-base/tree/trunc/hott/truncation`.

7 R. L. Constable, S. F. Allen, H. M. Bromley, W. R. Cleaveland, J. F. Cremer, R. W. Harper, D. J. Howe, T. B. Knoblock, N. P. Mendler, P. Panangaden, J. T. Sasaki, and S. F. Smith. *Implementing Mathematics with the NuPRL Proof Development System*. Prentice-Hall, NJ, 1986.

8 Nicolai Kraus. The general universal property of the propositional truncation. *ArXiv e-prints*, Nov 2014. To appear in the post-proceedings of TYPES'14.

9 Nicolai Kraus. *Truncation Levels in Homotopy Type Theory*. PhD thesis, School of Computer Science, University of Nottingham, Nottingham, UK, 2015.

10 Nicolai Kraus, Martín Escardó, Thierry Coquand, and Thorsten Altenkirch. Notions of anonymous existence in Martin-Löf type theory. Submitted, 2014.

11 Nicolai Kraus and Christian Sattler. Higher homotopies in a hierarchy of univalent universes. *ACM Transactions on Computational Logic (TOCL)*, 16(2):18:1–18:12, April 2015.

12 Peter LeFanu Lumsdaine. Weak omega-categories from intensional type theory. In *Typed Lambda Calculi and Applications (TLCA)*, pages 172–187. Springer-Verlag, 2009.

13 Michael Shulman. Univalence for inverse diagrams and homotopy canonicity. *Mathematical Structures in Computer Science*, pages 1–75, Jan 2015.

14 The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. `homotopytypetheory.org/book`, Institute for Advanced Study, first edition, 2013.

15 Benno van den Berg and Richard Garner. Types are weak ω-groupoids. *Proceedings of the London Mathematical Society*, 102(2):370–394, 2011.

Leaving the Nest: Nominal Techniques for Variables with Interleaving Scopes

Murdoch J. Gabbay[1], Dan R. Ghica[2], and Daniela Petrişan[3]

1 **Heriot-Watt University, U.K.**
2 **University of Birmingham, U.K.**
3 **Radboud University, The Netherlands**

Abstract

We examine the key syntactic and semantic aspects of a nominal framework allowing scopes of name bindings to be arbitrarily interleaved. Name binding (e.g. $\lambda x.M$) is handled by explicit name-creation and name-destruction brackets (e.g. $\langle xMx\rangle$) which admit interleaving. We define an appropriate notion of alpha-equivalence for such a language and study the syntactic structure required for alpha-equivalence to be a congruence. We develop denotational and categorical semantics for dynamic binding and provide a generalised nominal inductive reasoning principle. We give several standard synthetic examples of working with dynamic sequences (e.g. substitution) and we sketch some preliminary applications to game semantics.

1998 ACM Subject Classification D.3.1 Formal Definitions and Theory

Keywords and phrases nominal sets, scope, alpha equivalence, dynamic sequences

Digital Object Identifier 10.4230/LIPIcs.CSL.2015.374

1 Introduction and motivation

In the syntax of formal languages it is common to see names created by locally-scoped operators such as $\lambda a.(st)$ and $\forall a.(\phi \Rightarrow \psi)$. The binders λ and $\forall$ have scope from the left-hand (to the matching right-hand) and scope is determined at the site of binding in the structure of the term. However, dynamically-scoped binding is an often encountered phenomena occurring whenever resources are allocated and freed explicitly. The most common situation is that of memory in C-like languages, but this also applies to other resources such as opening and closing files or network sockets. In general, a physical resource will be handled using a name which can be used by the program. The choice of name, between the allocation and the release of the resource, is irrelevant, leading to notions similar to binding and α-equivalence, but more finely grained, to account for possible scope interleaving.

Nominal techniques [15, 12, 24] provide a state-of-the-art formalism for reasoning about abstract syntax with *statically-scoped* binding. However, existing techniques do not accommodate syntax with *dynamically-scoped* binding. This paper addresses this issue by introducing a syntactic notion of *dynamic sequences* and suitable denotational and categorical models.

In dynamic sequences, scope is managed by name-creation and name-destruction brackets 'create a' and 'destroy a', written as $\langle a$ and $a\rangle$, respectively. These may be interleaved and need not match up; $\langle a\langle ba\rangle b\rangle$, $\langle a\langle bb\rangle a\rangle$, and indeed just $\langle a$ and $a\rangle$ are perfectly valid sequences. In the special case of a well-matched name-creation/-destruction pair the theory specialises back to something that models nominal-style atoms-abstraction. For instance, $\langle aaa\rangle$, just as the nominal atoms-abstraction $[a]a$, models the α-equivalence behaviour of $\lambda a.a$.

24th EACSL Annual Conference on Computer Science Logic (CSL 2015).
Editor: Stephan Kreutzer; pp. 374–389

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

To define a mathematically well-behaved notion of α-equivalence, the basic notions of *free* and *bound* names require generalisation, and a notion of *freshness arity* emerges (Sec. 2), generalising the *freshness side-conditions* of nominal terms. In Sec. 3 we provide a relational semantics for dynamic sequences and in Sec. 4 we take stock of the monoid structure on dynamic sequences. We give an equational axiomatisation for a notion of *dynamic binding monoid*, that is, a monoid equipped with compatible nominal set structure and with left and right binders. We then prove that dynamic sequences form a free such dynamic binding monoid and obtain as a corollary a structural induction principle and a simpler characterisation for the α-equivalence relation.

Dynamic sequences have a 'flat' monoid structure, as opposed to the syntax trees one encounters in nominal abstract syntax. We use this 'flatness' to our advantage, as it allows us to interpret the non-hierarchical structure of interleaved scope. Thus one interpretation of dynamic sequences is as the structures used in game semantics. As an application, Sec. 6 gives a resource-sensitive formulation of pointer sequences as used in game semantics. We conclude in Sec. 7 with an overview of related work and directions for future research.

2 Dynamic sequences

2.1 Preliminaries

Let $\mathbb{A}$ be a countably infinite set of *names* or *atoms*. Given a bijection (permutation) $\pi : \mathbb{A} \to \mathbb{A}$ define its *support* by $\mathsf{supp}(\pi) = \{a \in \mathbb{A} \mid \pi a \neq a\}$. Write $\mathsf{Perm}(\mathbb{A})$ for the set of all permutations with finite support. Write ι for the *identity permutation* and $(a\ b)$ for the *swapping* or *transposition* of a and b.

If X is a set with a $\mathsf{Perm}(\mathbb{A})$-action, write this action infix as $-\cdot-$. An element $x \in X$ is *supported* by $A \subseteq \mathbb{A}$ when for all $\pi \in \mathsf{Perm}(\mathbb{A})$, $\forall a \in A.\pi a = a$ implies $\pi \cdot x = x$. Given $\pi \in \mathsf{Perm}(\mathbb{A})$ and $x \in X$, we say that π fixes x when $\pi \cdot x = x$. A *nominal set* is a set with a $\mathsf{Perm}(\mathbb{A})$-action where every element x has finite support. It is a fact that if X is a nominal set then every $x \in X$ has a least finite support, which we write $\mathsf{supp}(x)$.[1] We are interested in elements with finite support. If $a \in \mathbb{A}$ such that $a \notin \mathsf{supp}(x)$ we write $a\#x$. If X, Y are sets with $\mathsf{Perm}(\mathbb{A})$-action, call $f : X \to Y$ *equivariant* when $f(\pi \cdot x) = \pi \cdot (fx)$ for every $\pi \in \mathsf{Perm}(\mathbb{A})$ and $x \in X$. Finally, if $\phi(c)$ is a predicate on names, write $И c.\phi$ for "$\phi(c)$ holds for all but finitely many $c \in \mathbb{A}$"; this is the *NEW-quantifier* and we may read it as "for *fresh* c, $\phi(c)$". For more on the theory above see [15, 12, 24].

▶ **Definition 1.** Fix disjoint sets $\mathbb{A}$ of *atoms* and $\mathbb{K}$ of *constants*, writing $a, b, c \in \mathbb{A}$ and $k \in \mathbb{K}$. Define sets $\mathbb{T}$ of *tokens* and RSeq of *raw sequences*, writing $m \in \mathbb{T}$ and $e \in \mathsf{RSeq}$, inductively by: $m ::= a \mid a\rangle \mid \langle a \mid k$, $e ::= \varepsilon \mid em$.

RSeq is equivalently the set of lists of tokens and is the free monoid on $\mathbb{T}$. The set $\mathbb{K}$ can be equipped with a trivial $\mathsf{Perm}(\mathbb{A})$-action: every permutation fixes all the elements of $\mathbb{K}$. The permutation actions on $\mathbb{A}$ and $\mathbb{K}$ can be extended pointwise on the elements of raw sequences; for instance, $(c\ a)\cdot cc\rangle b\rangle a\rangle\langle c = aa\rangle b\rangle c\rangle\langle a$. Then RSeq is a nominal set and the support of a sequence is the set of names occurring in it. The set RSeq also has a monoid structure given by concatenation, which is compatible with the permutation action (monoid multiplication is equivariant and ε has empty support and thus is fixed by all permutations.)

[1] The set $\mathsf{Perm}(\mathbb{A})$ can also be seen as a nominal set with the $\mathsf{Perm}(\mathbb{A})$-action given by conjugation. The support of a permutation π is indeed the set $\mathsf{supp}(\pi)$ as defined in the previous paragraph.

Our notation and terminology suggest we should read the raw sequence $\langle aaa\rangle$ as "create a, use a (in some manner), then destroy it"—so if we assume a constant $\lambda \in \mathbb{K}$ then $\lambda\langle aaa\rangle$ shall, informally, model the syntax $\lambda a.a$.

We can call the binding of raw sequences, which we will make formal shortly, *dynamic* in the sense that scope is not determined by a single binder but by bracket-*pairs*; $\langle a$ does not 'know' where its matching $a\rangle$ is, or vice versa, and indeed $\langle a$ on its own and unpaired with any $a\rangle$ is also a valid raw sequence, as are $a\rangle$, $\langle aa\langle aaa\rangle$, and so forth.

We endow raw sequences with a binding structure using the following ideas, which will be illustrated in Ex. 6:

- *Bound:* An atom is *bound* if it is in the scope of a creation well-paired with a destruction.
- *Created:* An atom may appear following a creation operation which is not followed by a matching destruction.
- *Destructed:* Conversely, a destruction operation may appear without a matching creation.
- *Free:* An atom may be used without being created or destructed.

An atom occurrence cannot be characterised as merely 'free' or 'bound', but we need the more refined notion of *freshness arities*. We define a freshness arity as an element of a monoid $\mathcal{B}$, which we call the *binding monoid*, and which is the free monoid over carrier $\{\mathsf{c},\mathsf{f},\mathsf{d}\}$ modulo the following equations:

$$\mathsf{f}\cdot\mathsf{f} = \mathsf{f} \qquad \text{absorption} \qquad (1)$$

$$\mathsf{c}\cdot\mathsf{f} = \mathsf{c} \qquad \text{pre-absorption} \qquad (2)$$

$$\mathsf{f}\cdot\mathsf{d} = \mathsf{d} \qquad \text{post-absorption} \qquad (3)$$

$$\mathsf{c}\cdot\mathsf{d} = \varepsilon \qquad \text{cancellation} \qquad (4)$$

The freshness arity is assigned by a monoid homomorphism $\mathcal{F}_a : \mathsf{RSeq} \to \mathcal{B}$ defined by:

$$\mathcal{F}_a = \{\langle a \mapsto \mathsf{c}, \langle b \mapsto \varepsilon, a \mapsto \mathsf{f}, b \mapsto \varepsilon, a\rangle \mapsto \mathsf{d}, b\rangle \mapsto \varepsilon, k \mapsto \varepsilon\} \text{ where } (a \neq b \in \mathbb{A},\ k \in \mathbb{K}).$$

The set of finitely supported functions denoted by $\mathcal{B}^{\mathbb{A}}$ has a nominal monoid structure, with the multiplication of functions defined pointwise. The proof of the next lemmas is immediate.

▶ **Lemma 2.** *The map $\mathcal{F} : \mathsf{RSeq} \to \mathcal{B}^{\mathbb{A}}$ defined by $e \mapsto \lambda a.\mathcal{F}_a(e)$ is an equivariant monoid morphism.*

▶ **Lemma 3.** *For any $\beta \in \mathcal{B}$ there are unique $m, p \in \mathbb{N}$ and $n \in \{0, 1\}$ such that $\beta =_{\mathcal{B}} \mathsf{d}^m\cdot\mathsf{f}^n\cdot\mathsf{c}^p$.*

▶ **Definition 4.** Call the unique representation of $\beta \in \mathcal{B}$ its *normal form*.

▶ **Definition 5.** Given a sequence e and a name a it is helpful to introduce some notational shortcuts regarding the arity of a in e:

- We write $a \triangleright e$ when $\mathcal{F}_a e = \mathsf{d}^m\cdot\mathsf{f}^n$, for some $m, n \in \mathbb{N}$, i.e. there are no pending unmatched name creations $\langle a$ in e.
- We write $a \triangleleft e$ when $\mathcal{F}_a e = \mathsf{f}^n\cdot\mathsf{c}^m$, for some $m, n \in \mathbb{N}$, i.e. there are no pending unmatched destructors $a\rangle$ in e.
- We write $a \diamond e$, and call a *balanced* in e, when $a \triangleright e \wedge a \triangleleft e$. That is, there are no un-matched a-creations or a-destructions, so any occurrence of a is, informally, either 'bound' ($\mathcal{F}_a e = \varepsilon$) or 'free' ($\mathcal{F}_a e = \mathsf{f}$) in the conventional sense.

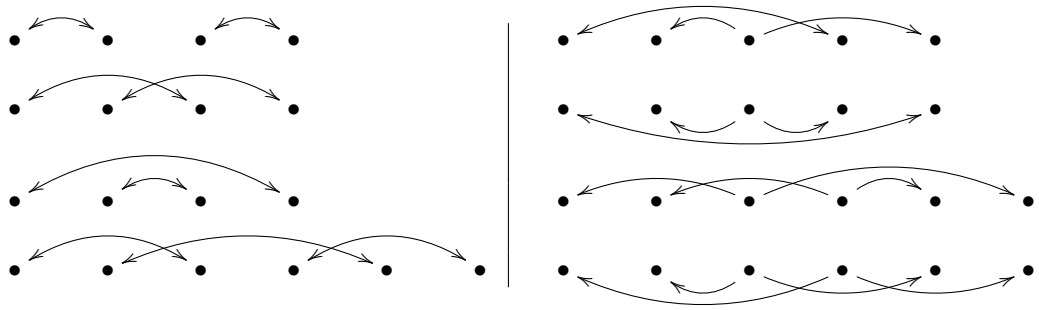


Figure 1 Pointer sequence illustration of dynamic sequences with $\mathcal{F}_a(e) = \varepsilon$.

▶ **Example 6.** Here are some 3-long sequences involving atom a, showing various arities.

$$\mathcal{F}_a(\langle a\langle a\langle a) = \mathsf{c}^3 \qquad \mathcal{F}_a(a\rangle\langle a\langle a) = \mathsf{dc}^2$$
$$\mathcal{F}_a(\langle aa\rangle a\rangle) = \mathsf{d} \qquad \mathcal{F}_a(a\rangle aa\rangle) = \mathsf{d}^2$$
$$\mathcal{F}_a(a\rangle a\rangle a) = \mathsf{d}^2\mathsf{f} \qquad \mathcal{F}_a(aaa) = \mathsf{f}$$
$$\mathcal{F}_a(\langle aa\rangle a) = \mathsf{f} \qquad \mathcal{F}_a(\langle aaa\rangle) = \varepsilon$$

2.2 α-equivalence

The pairing of atom creation and atom destruction operations creates a phenomenon similar to binding. The concrete choice of a name, between its creation and its destruction, should not matter. This leads directly to a dynamic version of the α-equivalence relation, illustrated by the following examples and non-examples.

$$\langle aa\rangle\langle bb\rangle =_\alpha \langle aa\rangle\langle aa\rangle \tag{5}$$
$$\langle a\langle ba\rangle b\rangle =_\alpha \langle b\langle cb\rangle c\rangle \tag{6}$$
$$\langle a\langle aa\rangle a\rangle =_\alpha \langle b\langle cc\rangle b\rangle \tag{7}$$
$$\langle a\langle ca\rangle\langle bc\rangle b\rangle =_\alpha \langle a\langle ca\rangle\langle ac\rangle a\rangle \tag{8}$$
$$\langle a\langle bba\rangle b\rangle \neq_\alpha \langle a\langle bbb\rangle a\rangle \tag{9}$$
$$\langle a\langle babb\rangle a\rangle \neq_\alpha \langle a\langle bbab\rangle a\rangle. \tag{10}$$

These sequences, in general sequences where $\mathcal{F}_a(e) = \varepsilon$ for any atom occurring in the sequence, can be informally illustrated using "pointer sequences": a node with a left-pointing arrow corresponds to a name creation, one with a right-pointing arrow to a name destruction, and an arrow-less dot to a name mention. Thus the pairs of α-equivalent sequences in (5)–(8) can be represented as the pointer sequences on the left column in Fig. 1, while the inequivalent pairs of sequences in (9) and (10) are represented on the right column in Fig. 1.

We now give a syntax-directed definition of α-equivalence.

▶ **Definition 7.** Define *alpha-equivalence* $=_\alpha \subseteq \mathsf{RSeq}\times\mathsf{RSeq}$ inductively by:

$$\frac{}{\varepsilon =_\alpha \varepsilon}\ (\alpha\varepsilon) \qquad \frac{e_1 =_\alpha e_2 \quad m \in \mathbb{T}}{e_1 m =_\alpha e_2 m}\ (\alpha\mathbf{m})$$

$$\frac{\textit{И}c.\, e_1\langle c\,(c\ a)\cdot e_2 =_\alpha e_1'\langle c\,(c\ b)\cdot e_2' \quad a \diamond e_2, b \diamond e_2'}{e_1\langle a e_2 a\rangle =_\alpha e_1'\langle b e_2' b\rangle}\ (\alpha\alpha)$$

(In $(\alpha\alpha)$ $(c\ a)\cdot e_2$ denotes the action of the permutation $(c\ a)$ on e_2, and similarly for $(c\ b)\cdot e_2'$.)

For other characterisations of $=_\alpha$ see Thm. 18 and Cor. 24.

The next two lemmas are instrumental in establishing that $=_\alpha$ is an equivalence relation and a congruence.

▶ **Lemma 8.** *If* $e_1 =_\alpha e_2$ *then* $\mathcal{F}(e_1) = \mathcal{F}(e_2)$.

▶ **Lemma 9.** $e_1\langle ae_2a\rangle{=}e_1'\langle ae_2'a\rangle$ *and* $a \diamond e_2, e_2'$ *imply* $e_1{=}e_1'$ *and* $e_2{=}e_2'$.

▶ **Lemma 10.** $=_\alpha$ *is an equivalence relation.*

Proof sketch. Transitivity is proved inductively using a case analysis of the possible rules ending the derivations. For example, assume that $ea\rangle =_\alpha ga'\rangle$ and $ga'\rangle =_\alpha ha''\rangle$ are both obtained using $(\alpha\alpha)$. Then we have $e = e_1\langle ae_2, g = g_1\langle a'g_2$ with $a \diamond e_2$, $a' \diamond g_2$ such that $И c.e_1\langle c\,(c\;a)\cdot e_2 =_\alpha g_1\langle c\,(c\;a')\cdot g_2$. Similarly, $g = g_1'\langle a'g_2', h = h_1\langle a''h_2$ with $a' \diamond g_2'$, $a'' \diamond h_2$ such that $И c.g_1'\langle c\,(c\;a')\cdot g_2' =_\alpha h_1\langle c\,(c\;a'')\cdot h_2$. Using Lem. 9 it follows that $g_i = g_i'$ from which we derive $И c.e_1\langle c\,(c\;a)\cdot e_2 =_\alpha h_1\langle c\,(c\;a'')\cdot h_2$ with $a \diamond e_2$, $a'' \diamond h_2$. Thus $ea\rangle =_\alpha ha''\rangle$ is obtained using $(\alpha\alpha)$. ◀

▶ **Theorem 11** ($=_\alpha$ is a congruence)**.** *If* $e_1 =_\alpha e_1'$ *and* $e_2 =_\alpha e_2'$ *then* $e_1e_2 =_\alpha e_1'e_2'$.

Proof sketch. By induction on e_2' with the only interesting case being when the second equivalence was obtained using $(\alpha\alpha)$. In this case, we have $e_2 = g_1\langle ag_2a\rangle$ and $e_2' = g_1'\langle bg_2'b\rangle$ such that $a \diamond g_2$, $b \diamond g_2'$ and $И c.\, g_1\langle c\,(c\;a)\cdot g_2 =_\alpha g_1'\langle c\,(c\;b)\cdot g_2'$. By the induction hypothesis we have $И c.\, e_1g_1\langle c\,(c\;a)\cdot g_2 =_\alpha e_1'g_1'\langle c\,(c\;b)\cdot g_2'$, hence by $(\alpha\alpha)$ we obtain $e_1e_2 =_\alpha e_1'e_2'$. ◀

▶ **Definition 12.** Let $\mathsf{DSeq} = (\mathsf{RSeq}/{=_\alpha})$ be the nominal set of sequences quotiented by α-equivalence, with the natural permutation action given by the action on representatives; call these *dynamic sequences*.

Note that Lem. 8 ensures that we can extend the notations of $a \diamond e, a \rhd e, a \lhd e$ to dynamic sequences.

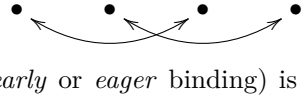
▶ **Lemma 13.** *If* $e \in \mathsf{DSeq}$ *and* $a \in \mathbb{A}$ *then* $\mathcal{F}_a e = \varepsilon$ *if and only if* $a\#e$.

2.3 On the congruence property of α-equivalence

The congruence of α-equivalence is an essential mathematical property which has motivated design decisions in our definition of dynamic scope. We take a moment to discuss them, and so perhaps gain a better perspective on the design space in which dynamic sequences exist. Consider the raw sequence $\langle aa\langle aaa\rangle$. Which of the two occurrences of $\langle a$ should match the destructor $a\rangle$? Rem. 9 and the equations of the binding monoid (1)–(4) uniquely identify it as the most recent unpaired $\langle a$ before the $a\rangle$ (so above, the rightmost $\langle a$ matches $a\rangle$). We call this *late binding*.

Some diagrams for the slightly more complex example of $\langle a\langle aa\rangle a\rangle$ illustrate this. We prefer the upper diagram to the lower diagram:

$\langle a\langle aa\rangle a\rangle =_\alpha \langle a\langle bb\rangle a\rangle =$ 

$\langle a\langle aa\rangle a\rangle \neq_\alpha \langle a\langle ba\rangle b\rangle =$ 

The lower diagram (which we might call *early* or *eager* binding) is not obviously mathematically wrong, but it is unreasonable in the sense that it invalidates congruence of α-equivalence:

▶ Remark. For any binding policy other than late binding, any reasonably defined $=_\alpha$ is not a congruence.

Informal argument. Whatever α-equivalence is, we require $\langle aa\rangle =_\alpha \langle bb\rangle$. If $=_\alpha$ is a congruence then $\langle a\langle aa\rangle =_\alpha \langle a\langle bb\rangle$. Given a binding policy which does not match a destructor for a with the most recent creation preceding it, it follows that $\langle a\langle aa\rangle =_\alpha \langle b\langle ab\rangle \neq_\alpha \langle a\langle bb\rangle$, a contradiction. ◀

Late binding preserves existing dynamic bindings whereas other binding policies do not, thus α-equivalence is a congruence with late binding (Thm. 11), whereas other dynamic binding policies are, in this sense, ill-behaved.

3 Relational semantics

We now give a concrete semantics in relations. This *relational semantics* is sound, complete, and compositional.

Call a *stack* a list of pairs of atoms, i.e., an element of the set $Stacks = (\mathbb{A}^2)^*$. Elements of $\mathbb{A}^2$ will be written as $(a{\mapsto}b)$. For each $S \in Stacks$ we define stack-like operations *read*, *add* and *remove*, written as $S(a)$, $S :: (a{\mapsto}b)$ and, respectively, $S\backslash a$. Both reading and removal of a record involve the most recent record $(a{\mapsto}b)$ in the stack. Formally, if $S = S_1 :: (a{\mapsto}b) :: S_2$ for stacks S_1, S_2, and $(a{\mapsto}c)$ does not occur in S_2 for any c, then $S(a) = b$ and $S\backslash a = S_1 :: S_2$. Otherwise $S(a)$ and $S\backslash a$ are undefined.

▶ **Definition 14.** Define a *relational semantics*

$$[\![\text{-}]\!] : \mathsf{RSeq} \to \mathcal{P}((Stacks \times (\mathbb{A} + \mathbb{K})^*)^2).$$

on raw sequences as follows:

$$[\![\varepsilon]\!] = \{((S,X),(S,X)) \mid \forall S, X\} \tag{11}$$
$$[\![ea]\!] = \{((S,X),(S',X' :: S'(a))) \mid ((S,X),(S',X')) \in [\![e]\!]\} \tag{12}$$
$$[\![ek]\!] = \{((S,X),(S',X' :: k)) \mid ((S,X),(S',X')) \in [\![e]\!]\} \tag{13}$$
$$[\![e\langle a]\!] = \{((S,X),(S' :: (a{\mapsto}b),X' :: b)) \mid ((S,X),(S',X')) \in [\![e]\!], b \# S', X'\} \tag{14}$$
$$[\![ea\rangle]\!] = \{((S,X),(S'\backslash a,X' :: S'(a))) \mid ((S,X),(S',X')) \in [\![e]\!]\} \tag{15}$$

In (12) and (15) it is assumed that $S'(a)$ and $S'\backslash a$ respectively are well-defined.

The intuition behind $(S,X)\,[\![e]\!]\,(S',X :: X')$ is quite operational. The stack S is to be thought of as a stack of name replacements, and the sequence X as a context in which e is interpreted. S' is an updated stack, since creation and destruction of names cause it to change and X' is a sequence which "interprets" e given the updated stack S' and the context X. Using a name a (see (12)) extends the current sequence with its stack value $S(a)$; creating a name $\langle a$ (see (14)) adds a new entry $(a{\mapsto}b)$ to the stack and extends the current sequence with b; destroying a name $a\rangle$ (see (15)) removes it from the stack and extends the current sequence with its dictionary value. A constant k is simply added to the sequence (see (13)).

The interpretation $[\![\text{-}]\!]$ is compositional, using pointwise relational composition $\text{-} \circ \text{-}$.

▶ **Lemma 15.** *For any sequence* $e \in \mathsf{RSeq}$ *and token* $m \in \mathbb{T}$: $[\![em]\!] = [\![e]\!] \circ [\![m]\!]$.

Proof. Immediate from definitions. ◀

▶ **Theorem 16.** *For any* $e, e' \in \mathsf{RSeq}$, $[\![ee']\!] = [\![e]\!] \circ [\![e']\!]$

Proof sketch. By Lemma 15 we have that for any token $m \in \mathbb{T}$ we have $[\![em]\!] = [\![e]\!] \circ [\![m]\!]$. Then we can use induction on the structure of e'. ◀

▶ **Proposition 17.** *If* $e_1, e_2 \in \mathsf{RSeq}$ *and* $m \in \mathbb{T}$ *then* $[\![em]\!] = [\![e'm]\!]$ *implies* $[\![e]\!] = [\![e']\!]$.

Proof. By Lem. 15 and simple calculations. ◀

▶ **Theorem 18.** *If* $e_1, e_2 \in \mathsf{RSeq}$ *then* $e_1 =_\alpha e_2$ *iff* $[\![e_1]\!] = [\![e_2]\!]$.

In view of soundness and completeness (Thm. 18) we can also use $[\![\text{-}]\!]$ to interpret dynamic sequences rather than raw sequences, i.e. $[\![\text{-}]\!] : \mathsf{DSeq} \to \mathcal{P}((\mathit{Stacks} \times (\mathbb{A} + \mathbb{K})^*)^2)$.

4 Equational axiomatisation

We give an equational axiomatisation of the interleaved dynamic binding of this paper (Def. 19). The dynamic sequences of Def. 12 form a free dynamic binding monoid (Thm. 23), and α-equivalence gets a purely equational characterisation as an equality subject to freshness-arity side-conditions (Cor. 24).

The central idea is to use a monoid structure equipped with a compatible permutation action, left and right 'binders' and a function with co-domain $\mathcal{B}^{\mathbb{A}}$ that encompasses the interleaved binding laws—which we will call the *freshness arity map*, see Def. 19 below. We will call such structures *dynamic binding monoids*.

Several approaches in the literature investigate notions of algebraic theories and equational reasoning in a nominal setting [14, 7, 6, 21]. A common denominator is that equations are presented with freshness side-conditions. For example, the η-rule in untyped λ-calculus can be captured by $a\#x \vdash \mathsf{lam}([a]\mathsf{app}(x, a)) = x$. An algebraic theory of dynamic binding monoids must interpret interleaved scope, and so the freshness side-conditions familiar from e.g. nominal unification, rewriting, and universal algebra must be suitably enriched to interpret freshness-*arity* side-conditions. Thus, some equations in Def. 19 have side-conditions on the freshness arity of variables specified using the binding monoid $\mathcal{B}$—if the reader prefers, these can also be seen as *typing* conditions.

▶ **Definition 19.** A *dynamic binding monoid* is a tuple $(M, ::, 1, \cdot, \langle, \rangle, \gamma)$ where $(M, \cdot)$ is a nominal set, $(M, ::, 1)$ is a monoid such that the binary operation is equivariant and 1 is an element of M with empty support, $\langle, \rangle : \mathbb{A} \to M$ are equivariant functions, and $\gamma : M \to \mathcal{B}^{\mathbb{A}}$ is an equivariant monoid morphism, satisfying equations:

$$\gamma_a(\langle a) = \mathsf{c}, \quad \gamma_a(a\rangle) = \mathsf{d}, \quad a\#x \vdash \gamma_a(x) = \varepsilon, \quad \text{and}$$
$$b\#m, \gamma_a(m) = \mathsf{f}^n \vdash \langle a :: m :: a\rangle = \langle b :: (b\ a)\cdot m :: b\rangle, \quad n \in \{0, 1\}. \tag{16}$$

Above, given $a \in \mathbb{A}$ we write $\gamma_a : M \to \mathcal{B}$ for the map $\lambda m.\gamma(m)(a)$, and $a\rangle$ for $\rangle$ at a (instead of $\rangle a$). We may omit the monoid multiplication :: when clear from the context.

▶ **Lemma 20.** *The set* DSeq *can be equipped with a dynamic binding monoid structure.*

A *morphism* between dynamic binding monoids $(M, ::, 1, \cdot, \langle, \rangle, \gamma)$ and $(M', ::, 1, \cdot, \langle', \rangle', \gamma')$ is an equivariant monoid morphism $f : M \to M'$ that preserves the left and right binders and the freshness arity map, that is, $f \circ \langle = \langle'$, $f \circ \rangle = \rangle'$, respectively $\gamma' \circ f = \gamma$.

Thus dynamic binding monoids form a category denoted by DBMon. Categories of nominal algebras described for example in [11, 21] have a forgetful functor to the category of underlying nominal sets Nom, and admit a free construction. That is, the forgetful functor

from a category of nominal algebras to Nom has a left adjoint. Such a free construction allows for deriving structural induction principles in the presence of binding, see for example [24, Cor. 8.22]. Because of the side-conditions involving the freshness arities, dynamic binding monoids lie outside the scope of nominal universal algebra. If we consider as the underlying structure of a binding monoid, not only the carrier nominal set, but also the freshness arity map, we can still obtain a free construction and hence a structural induction principle, see Thm. 22 below.

We introduce the category BNom of underlying nominal sets with a freshness arity map:

▶ **Definition 21.** Let BNom be the full subcategory of the slice category $\mathsf{Nom}/\mathcal{B}^{\mathbb{A}}$ with objects pairs (X,γ) where X is a nominal set and $\gamma : X \to \mathcal{B}^{\mathbb{A}}$ is an equivariant function such that $a\#x \vdash \gamma_a(x) = \varepsilon$.

A *morphism* in BNom from (X,γ) to (X',γ') is an equivariant function $f : X \to X'$ such that $\gamma' \circ f = \gamma$.

Let the *forgetful functor* $U : \mathsf{DBMon} \to \mathsf{BNom}$ send a dynamic binding monoid $(M, ::, 1, \cdot, \langle, \rangle, \gamma)$ to (M,γ).

▶ **Theorem 22.** *The forgetful functor* $U : \mathsf{DBMon} \to \mathsf{BNom}$ *has a left adjoint* F.

Proof. Consider $(X,\gamma) \in \mathsf{BNom}$, the set $X + \mathbb{A} + \mathbb{A} = X \cup \{\langle a | a \in \mathbb{A}\} \cup \{a\rangle | a \in \mathbb{A}\}$. We define an equivariant map $\overline{\gamma} : X + \mathbb{A} + \mathbb{A} \to \mathcal{B}^{\mathbb{A}}$ that acts as γ on X and such that $\overline{\gamma}_a(\langle a) = \mathsf{c}$, $\overline{\gamma}_a(a\rangle) = \mathsf{d}$, $\overline{\gamma}_a(\langle b) = \varepsilon$ and $\overline{\gamma}_a(b\rangle) = \varepsilon$ for $b \neq a$. Then $\overline{\gamma}$ can be extended uniquely to a monoid morphism $\gamma^* : (X + \mathbb{A} + \mathbb{A})^* \to \mathcal{B}^{\mathbb{A}}$. Define a relation $\equiv$ on $(X + \mathbb{A} + \mathbb{A})^*$ as the congruence generated by $\langle awa\rangle = \langle b(b\ a)\cdot wb\rangle$, where $a, b \in \mathbb{A}$, $w \in (X + \mathbb{A} + \mathbb{A})^*$, $b\#w$ and $\gamma^*_a(w) \in \{\varepsilon, \mathsf{f}\}$.

Construct $F(X,\gamma)$ as a dynamic binding monoid on the carrier nominal set $(X+\mathbb{A}+\mathbb{A})^*/\equiv$. Left and right binders are defined in the obvious way and the freshness arity map function is induced by γ^*. It is easy to check that whenever $w \equiv w'$ then $\gamma^*(w) = \gamma^*(w')$. We must exhibit an isomorphism $\mathsf{DBMon}((X + \mathbb{A} + \mathbb{A})^*/\equiv, M) \cong \mathsf{BNom}((X,\gamma),(M,\gamma_M))$. Starting with a morphism $f : X \to M$ in BNom, we can uniquely extend $f + \langle + \rangle : X + \mathbb{A} + \mathbb{A} \to M$ to an equivariant monoid morphism $f_\# : (X + \mathbb{A} + \mathbb{A})^* \to M$. We have that $\gamma \circ f_\# = \gamma^*$. It follows that for every $w, w' \in (X + \mathbb{A} + \mathbb{A})^*$ such that $w \equiv w'$ then $f_\#(w) = f_\#(w')$. Hence, $f_\#$ factors through a dynamic binding monoid morphism $\overline{f} : (X + \mathbb{A} + \mathbb{A})^*/\equiv \to M$. Conversely, given $g \in \mathsf{DBMon}((X + \mathbb{A} + \mathbb{A})^*/\equiv, M)$ we consider $g^\flat \in \mathsf{BNom}((X,\gamma),(M,\gamma_M))$ given by $g^\flat(x) = g([x])$ where $[x]$ is the $\equiv$-equivalence class of x. ◀

▶ **Theorem 23.** DSeq *is the free dynamic binding monoid on* $(\mathbb{A}\cup\mathbb{K}, \gamma_{\mathbb{A}})$ *where* $\gamma_{\mathbb{A}}(a)(a) = \mathsf{f}$, $\gamma_{\mathbb{A}}(a)(b) = \varepsilon$ *for* $b \neq a$ *and* $\gamma_{\mathbb{A}}(k)(a) = \varepsilon$.

Proof Sketch. We have that $\mathsf{DSeq} = \mathsf{RSeq}/=_\alpha$ and $\mathsf{RSeq} = (\mathbb{A}\cup\mathbb{K}+\mathbb{A}+\mathbb{A})^*$. Thus it suffices that $=_\alpha$ is equal to the relation $\equiv$ described in the proof of Thm. 22. That $\equiv \subseteq =_\alpha$ follows from the proof of Lem. 20. For the other inclusion, we prove by induction on the length of e that whenever $e =_\alpha e'$ then $e \equiv e'$. If the former equivalence was derived using the rules $(\alpha\varepsilon)$ or $(\alpha\mathbf{m})$ then the proof is immediate by induction. Assume that $e =_\alpha e'$ was derived using $(\alpha\alpha)$. That is, $e = e_1\langle ae_2a\rangle$, $e' = e'_1\langle be'_2b\rangle$ such that $a \diamond e_2$, $b \diamond e'_2$ and for any fresh c we have $e_1\langle c\,(c\ a)\cdot e_2 =_\alpha e'_1\langle c\,(c\ b)\cdot e'_2$. By inductive hypothesis $e_1\langle c\,(c\ a)\cdot e_2 \equiv e'_1\langle c\,(c\ b)\cdot e'_2$ for fresh c; we must prove $e_1\langle ae_2a\rangle \equiv e'_1\langle be'_2b\rangle$.

We consider the case when $e_1\langle c\,(c\ a)\cdot e_2 = h_1\langle dhd\rangle h_2$ and $e'_1\langle c\,(c\ b)\cdot e'_2 = h'_1\langle d'(d'\ d)\cdot hd'\rangle h'_2$ such that $h_1 \equiv h'_1$, $h_2 \equiv h'_2$, $d'\#h$ and $d \diamond h$. The most interesting case is when $h = g_1\langle cg_2$.

We have that

$$\begin{array}{lcl} e_1\langle ae_2a\rangle & = & h_1\langle dg_1\langle a(a\ c)\cdot g_2d\rangle(a\ c)h_2a\rangle \\ & \equiv & h_1\langle dg_1\langle cg_2d\rangle h_2c\rangle \\ & \equiv & h'_1\langle d'(d'\ d)\cdot g_1\langle c(d'\ d)\cdot g_2d'\rangle h'_2c\rangle \\ & \equiv & h'_1\langle d'(d'\ d)\cdot g_1\langle b(b\ c)(d'\ d)\cdot g_2d'\rangle(b\ c)h'_2b\rangle \\ & = & e'_1\langle be'_2b\rangle. \end{array}$$

It might be the case that $e_1\langle c\,(c\ a)\cdot e_2 \equiv e'_1\langle c\,(c\ b)\cdot e'_2$ was obtained using the transitivity of $\equiv$, for example $\langle d_1\langle d_2\langle cd_2\rangle d_1\rangle \equiv \langle d'_1\langle d'_2\langle cd'_2\rangle d'_1\rangle$ can only be derived using the transitivity of $\equiv$. For this case the proof that $e_1\langle ae_2a\rangle \equiv e'_1\langle be'_2b\rangle$ requires several steps and transitivity, but it reduces to the basic case resolved above. ◀

From the proof of Thm. 23 we obtain a new characterisation of $=_\alpha$ from Def. 7:

▶ **Corollary 24.** *The α-equivalence relation $=_\alpha$ on* RSeq *is the least congruence closed under the following rule ($a \diamond e$ is from Definition 5):*

$$\frac{b\#e \quad a \diamond e}{\langle aea\rangle =_\alpha \langle b\,(b\ a)\cdot eb\rangle}\ (\alpha).$$

Proof. From the proof of Thm. 23 it follows that α-equivalence on RSeq is equal to a relation $\equiv$, defined as the least congruence closed under the rule (α). ◀

5 Examples

In the examples to follow we see our formalism at work. These examples are elementary and, because of the way our framework is set up, the definitions are suitably simple.

Def. 12 defines a data type DSeq as the quotient of an inductive data type by an α-equivalence relation. We can reason on it by taking representatives of equivalence classes and working inductively on those representatives. This comes from how we define the set.

The reader familiar with nominal techniques might ask why we do not use *nominal abstract syntax* [15], where α-equivalence is built into the inductive definition of the data type at every inductive stage (using *atoms-abstraction*; a type constructor naturally present in the nominal universe), so we can work inductively up-to-α with no need for representatives.

That is impossible for us here because by design we do not know *a priori* when we write $\langle a$ in a dynamic sequence where (if anywhere) the matching $a\rangle$ will occur, and conversely, if we find $a\rangle$ in a dynamic sequence then we do not *a priori* know where (if anywhere) a matching $\langle a$ will occur.

Instead we will use a technique from [10] which allows us to lift function definitions from raw terms (in our case: raw sequences) to α-equivalence classes of raw terms (in our case: dynamic sequences). The required condition for this method to work is that what we call α-equivalence has the property of being *Barendregt-abstractive*; that every equivalence class must contain a member with maximum support. In our case it means that in the set of all raw sequences that represent the same dynamic sequence, there is one with a maximum number of atoms. We may call a representative of such an orbit a *Barendregt representative*, due to the intended similarity with the *Barendregt variable naming convention* [2]. Def. 25 will help us to calculate Barendregt representatives:

▶ **Definition 25.** Suppose $e \in$ RSeq is a raw sequence and $N \subseteq \mathbb{A}$ is a finite set of atoms. Define a function $\mathit{freshen}(N, e) :$ RSeq $\to$ RSeq inductively by:

- $\mathit{freshen}(N, \varepsilon) = \varepsilon$,
- $\mathit{freshen}(N, e_1\langle ae_2a\rangle) = \mathit{freshen}(N, e_1)\langle c\,\mathit{freshen}(N, (c\,a)\cdot e_2)\,c\rangle$ for a fresh $c \in \mathbb{A}$ (so $c\#e_1, e_2$ and $c \notin N$) provided that $a \diamond e_2$ holds, and
- $\mathit{freshen}(N, em) = \mathit{freshen}(N, e)m$ otherwise.

It is easy to check that $\mathit{freshen}(N, e)$ is well-defined, $\mathit{freshen}(N, e) =_\alpha e$, it has maximal support, and fresh atoms are distinct from N; the reason for this last condition will become clear in a moment.

It is convenient now to make a notational distinction between $e \in \mathsf{RSeq}$ and its equivalence class $[e]_\alpha \in \mathsf{DSeq}$. We have:

▶ **Lemma 26.** *Given $k, l \geq 0$, the map $\alpha(k,l) : (\mathsf{RSeq}^k \times \mathbb{A}^l) \to (\mathsf{DSeq}^k \times \mathbb{A}^l)$ defined by*

$$(e_1, \ldots, e_k, n_1, \ldots, n_l) \longmapsto ([e_1]_\alpha, \ldots, [e_k]_\alpha, n_1, \ldots, n_l)$$

is Barendregt-abstractive.

Proof. We must construct a Barendregt representative of $([e_1]_\alpha, \ldots, [e_k]_\alpha, n_1, \ldots, n_l)$. Write $N = \{n_1, \ldots, n_l\}$. Then we take $e'_1 = \mathit{freshen}(N, e_1)$, and $e'_2 = \mathit{freshen}(N \cup \mathsf{supp}(e'_1), e_2)$, and $e'_3 = \mathit{freshen}(N \cup \mathsf{supp}(e'_1) \cup \mathsf{supp}(e'_2), e_3)$, and so on. It clear that $(e'_1, \ldots, e'_k, n_1, \ldots, n_l)$ has maximal support and that it is a Barendregt representative of (the inverse image of) $([e_1]_\alpha, \ldots, [e_k]_\alpha, n_1, \ldots, n_l)$. Write this representative $\mathit{freshen}(e_1, \ldots, e_k, n_1, \ldots, n_l)$. ◀

▶ **Definition 27** ([22])**.** Suppose X and Y are nominal sets and $f : Y \to X$ is a function. Define $bv_f(y) = \mathsf{supp}(y) \setminus \mathsf{supp}(f(y))$.

For instance $bv_{\alpha(1,0)}(\langle aa\rangle) = \{a\}$.

Lem. 26 allows us to tailor [10, Thm. 27] to functions taking k dynamic sequences and l atoms as input (all our examples below will have this form) as follows:

▶ **Theorem 28.** *Suppose X is a nominal set and $k, l \geq 0$ and $F : \mathsf{RSeq}^k \times \mathbb{A}^l \to X$, and suppose for every $e_1, \ldots, e_k \in \mathsf{RSeq}$ and $n_1, \ldots, n_l \in \mathbb{A}$ that*

$$bv_{\alpha(k,l)}(\mathit{freshen}(e_1, \ldots, e_k, n_1, \ldots, n_l))\#F(\mathit{freshen}(e_1, \ldots, e_k, n_1, \ldots, n_l)).$$

Then the map $\mathcal{И}F : \mathsf{DSeq}^k \times \mathbb{A}^l \to X$ defined by

$$\mathcal{И}F([e_1]_\alpha, \ldots, [e_k]_\alpha, n_1, \ldots, n_l) = F(\mathit{freshen}(e_1, \ldots, e_k, n_1, \ldots, n_l))$$

is well-defined.

Proof. From Lem. 26 and [10, Thm. 27]. ◀

We specialise Thm. 28 to $k{=}l{=}1$ for illustration's sake: $\mathcal{И}F([e]_\alpha, n)$ is equal to $F(e', n)$ where bound atoms in e' are chosen distinct and not equal to n, and $\mathcal{И}F([\langle aa\rangle]_\alpha, a) = F(\langle bb\rangle, a)$. An equivalent phrasing of the condition in Thm. 28 is this:

$$\mathsf{supp}(F(\mathit{freshen}(e_1, \ldots, e_k, n_1, \ldots, n_l))) \subseteq \mathsf{supp}([e_1]_\alpha, \ldots, [e_k]_\alpha, n_1, \ldots, n_l).$$

When we use Thm. 28 we will tend to write $\mathcal{И}F$ just as F, thus, notationally identifying the function-on-α-equivalence-classes with the function-on-representatives. We obfuscate the distinction between e and $[e]_\alpha$ and write our definitions 'as if' they were by induction on dynamic sequences. Doing this is consistent with informal practice: for instance, we are used to writing $\mathit{size}(\lambda a.a)$ and saying "size of $\lambda a.a$" but actually meaning "pick a representative and calculate the size of that representative". Thus, the reader who cares about such things can unpick this obfuscation back to the raw sequences and maximally distinct representatives; the reader who does not care, should be able to read the text just as they would any 'inductive' definition on syntax quotiented by α-equivalence.

5.1 Operations on dynamic sequences

▶ **Example 29** (Counting name creation-destruction pairs)**.** Define a function $|\text{-}| : \mathsf{DSeq} \to \mathbb{N}$ using Thm. 28 by:

$$\begin{array}{rr} |\varepsilon| = 0 & a \rhd e \Rightarrow |ea\rangle| = |e|. \\ |e\langle a| = |e| & a \diamond e' \Rightarrow |e\langle ae'a\rangle| = |ee'| + 1 \\ |ek| = |e| & \end{array}$$

To apply Thm. 28 we must check that any maximally distinct choice of bound names in the argument is fresh for the result. This is indeed the case (since $a\#n$ for any atom and any $n{\in}\mathbb{Z}$). $|e|$ counts the number of pairs of matched creation-destructors in e. The side-conditions ensure that the clauses pick out the correct creation for each destructor. We calculate $|\text{-}|$ for an example sequence; in this example we mark instances of the atom a with subscripts to see how the answer is calculated (so a_1 and a_2 are the same atom; just different instances):

$$|\langle a_1 a_2 \langle a_3 a_4 a_5\rangle a_6\rangle| = |a_2\langle a_3 a_4 a_5\rangle| + 1 = |a_2 a_4| + 2 = 2.$$

In fact, the side-condition $a \diamond e'$ is superfluous, but it ensures that brackets are consumed in well-matched pairs.

▶ Remark. The clauses above actually define an inductive function on raw sequences. Thm. 28 gives us freshness-based conditions to verify that this induces a function on dynamic sequences (formally, $И|e|$).

Function $|e|$ happens to make sense for all raw sequences whether bound names are maximally distinct or not; for an example of where this not the case, see Ex. 31.

▶ **Example 30** (Counting bound occurrences)**.** Define a function $\|\text{-}\| : \mathsf{DSeq} \to \mathbb{N}$ using Thm. 28 as follows:

$$\begin{array}{rr} \|\varepsilon\| = 0 & a\#e' \Rightarrow \|e\langle ae'a\rangle\| = \|ee'\| \\ \|ek\| = \|e\| & a\#e', a \diamond e'' \Rightarrow \|e\langle ae'ae''a\rangle\| = \|e\langle ae'e''a\rangle\| + 1 \\ \|e\langle a\| = \|e\| & a \rhd e \Rightarrow \|ea\rangle\| = \|e\|. \end{array}$$

To apply Thm. 28 we must check that any maximally distinct choice of bound names in the argument is fresh for the result. This is indeed the case.

$\|ek\|$ counts how many names occur 'bound' in a dynamic sequence, i.e. between a matched pair of a creation and destructor. For example $\|a\langle aaa\rangle a\| = 1$ because there is only one occurrence of a between its creation and destruction. Two other occurrences of a are outside the scope. For the same example sequence as above we have:

$$\|\langle a_1 a_2 \langle a_3 a_4 a_5\rangle a_6\rangle\| = \|\langle a_1 \langle a_3 a_4 a_5\rangle a_6\rangle\|{+}1 = \|\langle a_3 a_4 a_5\rangle\|{+}1 = \|\langle a_3 a_5\rangle\|{+}2 = \|\varepsilon\|{+}2 = 2$$

The side-conditions ensure that brackets get 'eaten' in well-matched pairs, and are also used to identify the first free occurrence of the bound name.

▶ **Example 31** (Capture-avoiding substitution)**.** We define $\text{-}[\text{-}/\text{-}] : \mathsf{DSeq} \times \mathbb{A} \times \mathbb{A} \to \mathsf{DSeq}$ by:

$$\begin{array}{rr} \varepsilon[a/b] = \varepsilon & c{\neq}a \Rightarrow ec[a/b] = e[a/b]c \\ ek[a/b] = e[a/b]k & c{\neq}a \Rightarrow e\langle c[a/b] = e[a/b]\langle c \\ ea[a/b] = e[a/b]b & a \rhd e \Rightarrow ea\rangle[a/b] = e[a/b]b\rangle \\ e\langle a[a/b] = e[a/b]\langle b & И c.b \diamond e' \Rightarrow e\langle ce'c\rangle[a/b] = e[a/b]\langle c(e'[a/b])c\rangle \end{array}$$

Previous examples made sense on raw sequences even if bound names were not chosen maximally distinct, but this function uses more of the power of Thm. 28, licensing us in effect to rename bound atoms and so avoid accidental name-clashes: $\langle aa\rangle[a/b] = \langle cc\rangle[a/b] = \langle cc\rangle = \langle aa\rangle$.

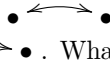
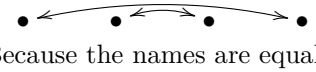
In fact, no rule above can be applied in $\langle aa\rangle[a/b]$, but we do not care because we only care about Barendregt representatives of triples (e, a, b) and such a representative will choose the bound names in e distinct from a and b. If we wish to notice that we are technically working with raw sequences and not dynamic sequences then we choose some junk value for the non-maximally-distinct cases.

For the final example we first introduce some notation, a regular-expression-like language for dynamic sequences.

▶ **Definition 32.** Define functions

$$+, \cdot : \mathcal{P}(\mathsf{DSeq}) \times \mathcal{P}(\mathsf{DSeq}) \to \mathcal{P}(\mathsf{DSeq})$$
$$-^* : \mathcal{P}(\mathsf{DSeq}) \to \mathcal{P}(\mathsf{DSeq}) \qquad \text{by}$$

$$e \in E + F \text{ iff } e \in E \text{ or } e \in F$$
$$e \in E \cdot F \text{ iff } \exists e_1, e_2 \in \mathsf{DSeq}. e = e_1 e_2 \text{ and } e_1 \in E, e_2 \in F$$
$$e \in E^* \text{ iff } e = \varepsilon \text{ or } e \in E \cdot E^*.$$

▶ **Example 33** (Capture-avoiding interleaving)**.** Another phenomenon resembling variable capture can occur when interleaving sequences. When we interleave a raw sequence such as $\langle aa\rangle$ with itself, we obtain the set of sequences $\{\langle aa\rangle\langle aa\rangle, \langle a\langle aa\rangle a\rangle\}$. If we represent them diagrammatically, we have the interleaving of  producing the sequences  and  . What happened to the sequence ? Because the names are equal the wrong creation is 'captured' by the wrong destructor in the course of interleaving. This can be avoided if we interleave the sequences up to α-equivalence.

Define $\| : \mathsf{DSeq} \times \mathsf{DSeq} \to \mathcal{P}(\mathsf{DSeq})$ using Thm. 28 and the notations from Def. 32:

$$\varepsilon \parallel e = e = e \parallel \varepsilon \qquad m\#e'm', m'\#em \Rightarrow em \parallel e'm' = (em \parallel e') \cdot m' + (e \parallel e'm') \cdot m.$$

To use Thm. 28 it suffices to check of each clause that maximally distinct choice of bound names do not affect the result, and we can assume that bound names in e_1 are chosen distinct from those in e_2, since this is a Barendregt representative of the input (e_1, e_2) to $\|$. Therefore

$$\langle aa\rangle \parallel \langle aa\rangle = \{\langle aa\rangle\langle aa\rangle, \langle a\langle aa\rangle a\rangle, \langle a\langle ba\rangle b\rangle\}$$

6 Application: Games with pointer sequences

In this section we sketch a potential application of dynamic sequences, a more formal and more resource-sensitive representation of pointer sequences in game semantics. Space restrictions prevent us from fully working this out but we hope it illustrates the potential of dynamic sequences to rigorously representat semantic models that require interleaved name scopes.

One of the original presentations of game semantics, by Hyland and Ong, represented *plays* as sequences of *moves* annotated with arrows between moves [18]. Formally, plays

were formalised as sequences equipped with a function f from natural numbers to natural numbers indicating that the move at index n points at move at index $f(n)$. [2]

Ghica and Gabbay [9] already gave a formulation of plays using raw sequences, which turned out to streamline key definitions and simplified many proofs. This paper did not consider α-equivalence and dynamic sequences, although some of our ideas are foreshadowed.

A pointer sequence is represented diagrammatically as an ordinary sequence decorated with pointed arrows, for example

would be represented using HO integer indices as the pair

$$(m_0 m_1 m_2 m_3 m_4 m_5, (1{\mapsto}0, 2{\mapsto}0, 3{\mapsto}2, 4{\mapsto}1, 5{\mapsto}3)).$$

The raw sequence representation is $m_0[a]* :: m_1[b]a :: m_2[c]a :: m_3[d]c :: m_4[e]b :: m_5[f]d$. Each move has a name, freshly introduced, indicated in square brackets, serving as an address, and it uses a previously introduced name to indicate the point of the arrow.

Game semantics requires complex operations on pointer sequences, such as swapping moves (while preserving pointers) to model reordering of actions in asynchronous concurrency [16] or extracting sub-sequences to model restricted history sensitiveness in languages without effects (*innocence* [18]). With integer indices, the pointer map needs to be re-indexed, an awkward operation which can be formalised in principle but it never was in practice due to sheer tedium. Using names, the same definitions are straightforward, as the names stay attached to the moves, making a more precise formalisation possible.

Although the raw sequence formalisation is from a mathematical point of view effective, from a conceptual and operational point of view is too profligate in its use of names. This is best illustrated with a simple example. The standard interpretation of the sequencing operator in HO games for ALGOL-like languages [1] is the set of even-length prefixes of this pointer sequence: 

. The operational intuition is as follows, where P is 'the program' and E is 'the environment':

r E asks P to start sequencing the two commands;

r_1 P in reply to r (see arrow) asks E to execute the first command;

d_1 E in reply to r_1 (see arrow) eventually reports the first command's termination;

r_2 P, justified by r (see arrow), asks E to execute the second command;

d_2 E, in reply to r_2, eventually reports the first command's termination;

d P, in reply to r, reports that sequencing is completed.

The raw sequence representation of this play is $\mathsf{r}[a]* :: \mathsf{r}_1[b]a :: \mathsf{d}_1 b :: \mathsf{r}_2[d]a :: \mathsf{d}_2 d :: \mathsf{d}a$, which requires 3 names. Certain moves, called *answers*, are never pointed at, so they need not introduce a name (they can, but it will simply be wasted). In certain game models answers have the additional property that after they point to a move no other subsequent move can ever point to it either—like name destructors, in fact! Using dynamic sequences with explicit name creation and destruction and late binding, the same sequence can be represented as: $\mathsf{r}*\langle a :: \mathsf{r}_1 a\langle a :: \mathsf{d}_1 a\rangle :: \mathsf{r}_2 a\langle a :: \mathsf{d}_2 a\rangle :: \mathsf{d}a\rangle$. The raw sequence representative of the dynamic sequence above uses just the name a. This is more aesthetically pleasing but it is also helpful for two reasons related to representing game models for program verification

[2] This section is best understood by readers familiar with Hyland-Ong-style game semantics, but it is written so that it can be also accessible to the casual reader.

or compilation: Dynamic sequences allow an improvement of the mathematical presentation of game semantics as well, beyond the raw-sequence formalisation. For example, in [9]:

- Def. 2.12, formalises the concept of "enabled sequence" e, in which any name of a move must be introduced before being used. This definition becomes just $a \lhd e$.
- Def. 2.15(iii) "*Call a play* e strictly scoped *when* $\mathsf{a}[b]e' \in e$ *implies* $a \notin \mathsf{supp}(e')$." says that once the "answer" move a uses a name a, that name should never be used again. This condition can be removed now and plays be made strictly scoped by construction, because a can be destructed by the answer move: $\mathsf{a}a\rangle\langle b :: e'$.

Note that, as detailed in Sec. 5, the various raw-sequence operations used in game semantics can be lifted to dynamic sequences in an elegant way.

7 Related and future work

The main contribution of this paper is the syntactic notion of *dynamic sequence* that models *interleaved* scope by splitting binding into two more primitive syntactic constructs: a *name-creation* bracket $\langle a$, and a *name-destruction* bracket $a\rangle$. By *interleaved* we mean that brackets need not be perfectly nested, as in $\langle a\langle baba\rangle b\rangle$.

The idea of splitting local binding into two brackets has been seen before. The Adbmal syntax from [17] splits λ-binding specifically in the λ-calculus into an opening bracket λa and a closing bracket $\curlywedge a$. However that paper is focused on scope-balanced terms and assumes a *jump* semantics, that is, $\curlywedge a$ closes the scope of all intermediate λs occurring before the matching λa in order to avoid interleaved scope. By contrast, in this paper $a\rangle$ lazily matches only the single most recent unmatched $\langle a$. It would be interesting to develop a categorical semantics for the $\curlywedge$-calculus and to explore further connections with dynamic sequences. It would be certainly interesting to extend the ideas of dynamic scope to trees, as Adbmal is set up to do, but this presents significant conceptual challenges, even before considering the technical ones. For example, matching brackets in a non-linear structure seems to require a notion of traversal for the structure. This remains to be investigated.

Dynamic scope also appears in natural languages, in semantic models for indefinite articles [28]. An opening bracket corresponds to the creation of a new 'file' for storing subsequent information and anchoring references. A closing bracket corresponds to the deletion of the 'file' and the destruction of the context. That paper takes a radically different approach based on a variation of monoidal categories and Grothendieck constructions. Working out the precise connection with our setting is left as future work.

In Sec. 5 we introduced a number of concepts such as regular expressions over dynamic sequences (Def. 32). Regular expressions and Kleene algebras with statically scoping nominal-style name-binding and -generation have been studied [13, 23, 20] and it would be interesting to investigate versions with dynamic scope. Languages with allocation have been extensively studied, including in the nominal setting (e.g. [3, 25]), but those with deallocation not so much as far as we know. This too could be future work. It would also be interesting to extend nominal automata [26, 4, 19] to handle name destruction. We could then investigate whether dynamic binding monoids play a similar role in understanding the algebraic theory of languages accepted by such automata, just as orbit-finite nominal monoids do for nominal languages, see [5].

Our original motivation was to apply dynamic sequences as a notation for the pointer sequences of game semantics, to simplify the formalisation of definitions of operations on pointer sequences and proofs of their properties. Raw nominal sequences are a step in this direction [9]; dynamic sequences take this further by introducing appropriate rules

for scope and α-equivalence. This may help to formalise parts of game semantics—think "game semantics in Nominal Isabelle" analogously to the current extensive implementation of nominal abstract syntax in Isabelle [27], or "rewriting on game semantics using nominal rewriting" (or a suitable generalisation with freshness side-conditions generalised to freshness-arity side-conditions) similar to [8]—and to tighten the connection between the game-semantic and abstract-machine models.

Perhaps the most significant challenge, but also the most exciting opportunity, is the use of dynamic sequences to model explicit resource management in C-like languages. Intuitively, a call to `malloc()` introduces a new name for a memory location, which in a dynamic trace corresponds to $\langle a$, whereas a call to `free()` removes that name, which in a dynamic trace corresponds to $a\rangle$. Clearly the scopes of the memory locations thus managed can be arbitrarily interleaved. However, the nominal aspects are only one aspect required to understand `malloc/free`. The stateful effects, the possibility of dangling pointers and garbage require significant amounts of further work. To conclude, we believe that interleaved name scopes are an interesting phenomenon which appears in several contexts: game semantics (our initial motivation), natural languages and low-level languages with explicit resource management. However, beyond these actual and potential applications, dynamic sequences seem to also be a novel nominal phenomenon, interesting in its own right.

Acknowledgements. We thank Bertram Wheen for a partial Agda formalisation of the syntactic model.

References

1 Samson Abramsky and Guy McCusker. Linearity, sharing and state: a fully abstract game semantics for idealized algol with active expressions. *Electr. Notes Theor. Comput. Sci.*, 3:2–14, 1996.

2 Henk P. Barendregt. *The Lambda Calculus: its Syntax and Semantics (revised ed.)*. North-Holland, 1984.

3 Nick Benton and Benjamin Leperchey. Relational reasoning in a nominal semantics for storage. In *Typed Lambda Calculi and Applications*, pages 86–101. Springer, 2005.

4 Mikolaj Bojanczyk, Bartek Klin, and Slawomir Lasota. Automata theory in nominal sets. *Logical Methods in Comp. Sci.*, 10(3), 2014.

5 Mikołaj Bojańczyk. Nominal monoids. *Theory of Computing Systems*, 53(2):194–222, 2013.

6 Ranald Clouston. Nominal lawvere theories: A category theoretic account of equational theories with names. *J. Comput. Syst. Sci.*, 80(6):1067–1086, 2014.

7 Ranald A. Clouston and Andrew M. Pitts. Nominal equational logic. *Electr. Notes Theor. Comput. Sci.*, 172:223–257, 2007.

8 Maribel Fernández and Murdoch J. Gabbay. Nominal rewriting (journal version). *Information and Computation*, 205(6):917–965, June 2007.

9 Murdoch Gabbay and Dan R. Ghica. Game semantics in the nominal model. *Electr. Notes Theor. Comput. Sci.*, 286:173–189, 2012.

10 Murdoch J. Gabbay. A general mathematics of names. *Information and Computation*, 205(7):982–1011, July 2007.

11 Murdoch J. Gabbay. Nominal algebra and the HSP theorem. *Journal of Logic and Computation*, 19(2):341–367, April 2009.

12 Murdoch J. Gabbay. Foundations of nominal techniques: logic and semantics of variables in abstract syntax. *Bulletin of Symbolic Logic*, 17(2):161–229, 2011.

13 Murdoch J. Gabbay and Vincenzo Ciancia. Freshness and name-restriction in sets of traces with names. In *FOSSACS*, volume 6604 of *Lec. Notes in Comp. Sci.*, pages 365–380. Springer, 2011.

14 Murdoch J. Gabbay and Aad Mathijssen. Nominal universal algebra: equational logic with names and binding. *Journal of Logic and Computation*, 19(6):1455–1508, December 2009.

15 Murdoch J. Gabbay and Andrew M. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects of Computing*, 13(3–5):341–363, July 2001.

16 Dan R. Ghica and Andrzej S. Murawski. Angelic semantics of fine-grained concurrency. *Ann. Pure Appl. Logic*, 151(2-3):89–114, 2008.

17 Dimitri Hendriks and Vincent van Oostrom. adbmal. In *CADE*, volume 2741 of *Lec. Notes in Comp. Sci.*, pages 136–150. Springer, 2003.

18 J. M. E. Hyland and C.-H. Luke Ong. On full abstraction for PCF: i, ii, and III. *Inf. Comput.*, 163(2):285–408, 2000.

19 Dexter Kozen, Konstantinos Mamouras, Daniela Petrisan, and Alexandra Silva. Nominal Kleene coalgebra. In M. M. Halldórsson, K. Iwama, N. Kobayashi, and B. Speckmann, editors, *Proc. 42nd Int. Colloq. Automata, Languages, and Programming, Part II (ICALP 2015)*, volume 9135 of *Lecture Notes in Computer Science*, pages 290–302, Kyoto, Japan, July 6–10 2015. EATCS, Springer.

20 Dexter Kozen, Konstantinos Mamouras, and Alexandra Silva. Completeness and incompleteness in nominal Kleene algebra. Technical report, Computing and Information Science, Cornell University, November 2014.

21 Alexander Kurz and Daniela Petrisan. On universal algebra over nominal sets. *Math. Struc. in Comp. Sci.*, 20(2):285–318, 2010.

22 Alexander Kurz, Daniela Petrisan, Paula Severi, and Fer-Jan de Vries. Nominal coalgebraic data types with applications to lambda calculus. *Logical Methods in Computer Science*, 9(4), 2013.

23 Alexander Kurz, Tomoyuki Suzuki, and Emilio Tuosto. On nominal regular languages with binders. In *FOSSACS*, volume 7213 of *Lec. Notes in Comp. Sci.*, pages 255–269. Springer, 2012.

24 Andrew M Pitts. *Nominal Sets: Names and Symmetry in Comp. Sci.*, volume 57. Cambridge University Press, 2013.

25 Mark R Shinwell and Andrew M Pitts. On a monadic semantics for freshness. *Theor. Comp. Sci.*, 342(1):28–55, 2005.

26 Nikos Tzevelekos. Fresh-register automata. In *POPL*, pages 295–306. ACM, 2011.

27 Christian Urban. Nominal reasoning techniques in Isabelle/HOL. *J. of Automatic Reasoning*, 40(4):327–356, 2008.

28 Albert Visser, Kees Vermeulen, et al. Dynamic bracketing and discourse representation. *Notre Dame Journal of Formal Logic*, 37(2):321–365, 1996.

Rank Logic is Dead, Long Live Rank Logic!

Erich Grädel and Wied Pakusa

Mathematical Foundations of Computer Science, RWTH Aachen University, Germany
{graedel,pakusa}@logic.rwth-aachen.de

Abstract

Motivated by the search for a logic for polynomial time, we study rank logic (FPR) which extends fixed-point logic with counting (FPC) by operators that determine the rank of matrices over finite fields. While FPR can express most of the known queries that separate FPC from PTIME, nearly nothing was known about the limitations of its expressive power.

In our first main result we show that the extensions of FPC by rank operators over different prime fields are incomparable. This solves an open question posed by Dawar and Holm and also implies that rank logic, in its original definition with a distinct rank operator for every field, fails to capture polynomial time. In particular we show that the variant of rank logic FPR* with an operator that uniformly expresses the matrix rank over finite fields is more expressive than FPR.

One important step in our proof is to consider solvability logic FPS which is the analogous extension of FPC by quantifiers which express the solvability problem for linear equation systems over finite fields. Solvability logic can easily be embedded into rank logic, but it is open whether it is a strict fragment. In our second main result we give a partial answer to this question: in the absence of counting, rank operators are strictly more expressive than solvability quantifiers.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes, F.4.1 Mathematical Logic

Keywords and phrases logic, descriptive complexity, polynomial time, rank logic

Digital Object Identifier 10.4230/LIPIcs.CSL.2015.390

1 Introduction

"Le roi est mort, vive le roi!" has been the traditional proclamation, in France and other countries, to announce not only the death of the monarch, but also the immediate installment of his successor on the throne. The purpose of this paper is to kill the rank logic FPR, in the form in which it has been proposed in [7], as a candidate for a logic for PTIME. The logic FPR extends fixed-point logic by operators rk_p (for every prime p) which compute the rank of definable matrices over the prime field $\mathbb{F}_p$ with p elements. Although rank logic is well-motivated, as a logic that strictly extends fixed-point logic with counting by the ability to express important properties of linear algebra, most notably the solvability of linear equation systems over finite fields, our results show that the choice of having a separate rank operator for every prime p leads to a significant deficiency of the logic. Indeed, it follows from our main theorem that even the uniform rank problem, of computing the rank of a given matrix over an arbitrary prime, cannot be expressed in FPR and thus separates FPR from PTIME. This also reveals that a more general variant of rank logic, which has already been proposed in [14, 15, 17] and which is based on a rank operator that takes not only the matrix but also the prime p as part of the input, is indeed strictly more powerful than FPR. Our result thus installs this new rank logic, denoted FPR*, as the rightful and distinctly more powerful successor of FPR as a potential candidate for a logic for PTIME. A full version of this paper can be found at [11].

24th EACSL Annual Conference on Computer Science Logic (CSL 2015).
Editor: Stephan Kreutzer; pp. 390–404

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

A logic for polynomial time. The question whether there is a logic that expresses precisely the polynomial-time properties of finite structures is an important challenge in the field of finite model theory [10, 12]. The logic of reference for this quest is fixed-point logic with counting (FPC) which captures polynomial time on many interesting classes of structures and which is strong enough to express many of the fundamental techniques which are used in polynomial-time algorithms [5]. Although it has been known for more than twenty years that FPC fails to capture PTIME in general, by the fundamental CFI-construction due to Cai, Fürer, and Immerman [4], we still do not know many properties of finite structures that provably separate FPC from PTIME. The two main sources of such problems are tractable cases of the graph isomorphism problem and queries from the field of linear algebra. First of all, the CFI-construction shows that FPC cannot define the isomorphism problem on graphs with bounded degree *and* bounded colour class size whereas the isomorphism problem is known to be tractable on all classes of graphs with bounded degree *or* bounded colour class size. Secondly, Atserias, Bulatov and Dawar [2] proved that FPC cannot express the solvability of linear equation systems over any finite Abelian group. It follows, that also other problems from the field of linear algebra are not definable in FPC. Interestingly, also the CFI-query can be formulated as linear equation system over $\mathbb{F}_2$ [7].

Rank logic. This latter observation motivated Dawar, Grohe, Holm and Laubner [7] to introduce *rank logic* (FPR) which is the extension of FPC by operators for the rank of definable matrices over prime fields $\mathbb{F}_p$. To illustrate the idea of rank logic, let $\varphi(x,y)$ be a formula (of FPC, say) which defines a binary relation $\varphi^{\mathfrak{A}} \subseteq A \times A$ in an input structure $\mathfrak{A}$. We identify the relation $\varphi^{\mathfrak{A}}$ with the associated adjacency matrix

$$M_\varphi^{\mathfrak{A}} : A \times A \to \{0,1\}, (a,b) \mapsto \begin{cases} 1, & \text{if } (a,b) \in \varphi^{\mathfrak{A}} \\ 0, & \text{if } (a,b) \notin \varphi^{\mathfrak{A}}. \end{cases}$$

In this sense, the formula φ defines in every structure $\mathfrak{A}$ a matrix $M_\varphi^{\mathfrak{A}}$ with entries in $\{0,1\} \subseteq \mathbb{F}_p$. Now, rank logic FPR provides for every prime $p \in \mathbb{P}$ a *rank operator* rk_p which can be used to form a *rank term* $[\mathsf{rk}_p\, \varphi(x,y)]$ whose value in an input structure $\mathfrak{A}$ is the matrix rank of M_φ over $\mathbb{F}_p$ (we remark that rank logic also allows to express the rank of matrices which are indexed by tuples of elements; the precise definition is given in Section 2).

It turns out that rank operators have quite surprising expressive power. For example, they can define the transitive closure of symmetric relations, they can count the number of paths in DAGs modulo p and they can express the solvability of linear equation systems over finite fields (recall that a linear equation system $M \cdot \vec{x} = \vec{b}$ is solvable if, and only if, $\mathsf{rk}(M) = \mathsf{rk}(M \,|\, \vec{b})$) [7]. Furthermore, rank operators can be used to define the isomorphism problem on various classes of structures on which the Weisfeiler-Lehman method (and thus fixed-point logic with counting) fails, e.g. classes of C(ai)-F(ürer)-I(mmerman) graphs [4, 7] and multipedes [13, 14]. The common idea of these isomorphism procedures is to reduce the isomorphism problem of structures to a suitable linear equation system over a finite field. More generally, by a recent result (which is mainly concerned with another candidate of a logic for polynomial time [1]), it follows that FPR captures polynomial time on certain classes of structures of bounded colour class size. In particular, this holds for the class of all structures of colour class size two (to which CFI-graphs and multipedes belong).

While these results clearly show the high potential of rank logic, almost nothing has been known about its limitations. For instance, it has remained open whether rank logic suffices to capture polynomial time, whether rank operators can simulate fixed-point inductions [7]

and also whether rank logic can define closely related problems from linear algebra (such as the solvability of linear equations over finite *rings* rather than fields [6]). One particular intriguing question is whether rank operators over different prime fields can simulate each other. In other words: is it possible to reduce the problem of determining the rank of a matrix over $\mathbb{F}_p$ (within fixed-point logic with counting) to the problem of determining the rank of a matrix over $\mathbb{F}_q$ (where p, q are distinct primes)? To attack this problem, Dawar and Holm [8, 14] developed a powerful toolkit of so called *partition games* of which one variant (so called *matrix-equivalence games*) precisely characterises the expressive power of infinitary logic extended by rank quantifiers. By using these games, Holm [14] was able to give a negative answer to the above question for the restricted case of rank operators of dimension one.

In this paper we propose a different method, based on exploiting symmetries rather than game theoretic arguments, to prove new lower bounds for logics with rank operators. In our main result (Theorem 3) we prove that for every prime q there exists a class of structures $\mathcal{K}_q$ on which FPC fails to capture polynomial time and on which rank operators over *every* prime field $\mathbb{F}_p$, $p \neq q$ can be simulated in FPC. On the other hand, rank operators over $\mathbb{F}_q$ can be used to canonise structures in $\mathcal{K}_q$ which means that the extension of fixed-point logic by rk_q-operators captures polynomial time on $\mathcal{K}_q$. From this result we can easily extract the following answers to the open questions outlined above.

(a) Rank logic (as defined in [7]) fails to capture polynomial time (Theorem 2).

(b) The extensions of fixed-point logic by rank operators over different prime fields are incomparable (Theorem 1), cf. [14, 8, 15].

We obtain these classes of structures $\mathcal{K}_q$ by generalising the well-known construction of Cai, Fürer and Immerman [4]. It has been observed that their construction actually is a clever way of encoding a linear equation system over $\mathbb{F}_2$ into an appropriate graph structure (see e.g. [2, 7, 14, 15]). Intuitively, each gadget in the CFI-construction can be seen as an equation (or, equivalently, as a circuit gate) which counts the number of transpositions of adjacent edges modulo two, and the CFI-query is to decide whether the total number of such transpositions is even or odd. Knowing this, it is very natural to ask whether this idea can be generalised to encode linear equation systems over arbitrary finite fields or, more generally, equation systems over arbitrary (Abelian) groups.

In [18], in order to obtain hardness results for the graph isomorphism problem, Torán followed this idea and established a graph construction which simulates mod k-counting gates for all $k \geq 2$. Moreover, in order to separate the fragments of rank logic by operators over different prime fields, Holm presented in [14] an even more general kind of construction which allows the representation of equations over *every* Abelian group G. In fact, we obtain the classes $\mathcal{K}_q$ essentially by using his construction for the special case where $G = \mathbb{F}_q$.

Solvability logic. One important step in our proof is to consider *solvability logic* FPS which is the extension of FPC by quantifiers which can express the solvability of linear equation systems over finite fields (so called *solvability quantifiers*, see [6, 17]). Obviously the logic FPS can easily be embedded into rank logic (as rank operators can be used to solve linear equation systems), but it remains open whether the inclusion $\mathrm{FPS} \leq \mathrm{FPR}$ is strict. To prove our main result outlined above we show that over certain classes of structures the logics FPS and FPR have precisely the same expressive power. In a more general context this might give some evidence that in the framework of fixed-point logic with counting rank operators can be simulated by solvability quantifiers. On the other hand we show in Section 4 that the extension of first-order logic (without counting) by solvability quantifiers is strictly weaker

than the respective extension by rank operators. This last result thus separates solvability quantifiers and rank operators in the absence of counting.

2 Logics with linear-algebraic operators

By $\mathcal{S}(\tau)$ we denote the class of all *finite, relational* structures of signature τ. We assume that the reader is familiar with *first-order logic* (FO) and *inflationary fixed-point logic* (FP). For details see [9, 10]. We write $\mathbb{P}$ for the set of primes and denote the prime field with p elements by $\mathbb{F}_p$. We consider matrices and vectors over *unordered* index sets. Formally, if I and J are non-empty sets, then an $I \times J$-matrix M over $\mathbb{F}_p$ is a mapping $M : I \times J \to \mathbb{F}_p$ and an I-vector $\vec{v}$ over $\mathbb{F}_p$ is a mapping $\vec{v} : I \mapsto \mathbb{F}_p$.

A *preorder* $\preceq$ on A is a reflexive, transitive and total binary relation. It induces a linear order on the classes of the associated equivalence relation $x \sim y := (x \preceq y \wedge y \preceq x)$. We write $A = C_0 \preceq \cdots \preceq C_{n-1}$ to denote the decomposition of A into $\sim$-classes C_i which are linearly ordered by $\preceq$ as indicated. We denote by $\mathrm{Aut}(\mathfrak{A}) \leq \mathrm{Sym}(A)$ the automorphism group of a structure $\mathfrak{A}$ as a subgroup of the symmetric group acting on the set A. We assume that the reader is familiar with the basic notions from (linear) algebra.

We recall the definitions of *first-order logic with counting* FOC and *(inflationary) fixed-point logic with counting* FPC. Formulas of FOC and FPC are evaluated over the *two-sorted extension* of an input structure by a copy of the arithmetic. Following [7] we let $\mathfrak{A}^{\#}$ denote the two-sorted extension of a τ-structure $\mathfrak{A} = (A, R_1, \ldots, R_k)$ by the arithmetic $\mathfrak{N} = (\mathbb{N}, +, \cdot, 0, 1)$, i.e. the two-sorted structure $\mathfrak{A}^{\#} = (A, R_1, \ldots, R_k, \mathbb{N}, +, \cdot, 0, 1)$ where the universe of the first sort (also referred to as *vertex sort*) is A and the universe of the second sort (also referred to as *number sort* or *counting sort*) is $\mathbb{N}$.

As usual for the two-sorted setting we have typed first-order variables, where Latin letters $x, y, z, \ldots$ stand for variables that range over vertices, and Greek letters $\nu, \mu, \ldots$ for variables ranging over numbers. For second-order variables we allow mixed types, i.e. a relation symbol R of type $(k, \ell) \in \mathbb{N} \times \mathbb{N}$ stands for a relation $R \subseteq A^k \times \mathbb{N}^\ell$. Of course, already first-order logic over such two-sorted extensions is undecidable. To obtain logics whose data complexity is in polynomial time we restrict the quantification over the number sort by a numeric term t, i.e. $Q\nu \leq t.\varphi$ where $Q \in \{\exists, \forall\}$ and where t is a closed *numeric* term. Similarly, for fixed-point logic FP we bound the numeric components of fixed-point variables R of type (k, ℓ) in all fixed-point definitions $\left[\mathsf{ifp}\, R\bar{x}\bar{\nu} \leq \bar{t}\,.\,(\varphi(\bar{x}, \bar{\nu}))\right](\bar{x}, \bar{\nu})$ by a tuple of closed numeric terms $\bar{t} = (t_1, \ldots, t_\ell)$ where each t_i bounds the range of the variable ν_i in the tuple $\bar{\nu}$. For the logics which we consider here the value of such numeric terms (and thus the range of all quantifiers over the number sort) is polynomially bounded in the size of the input structure. Together with the standard argument that inflationary fixed-points can be evaluated in polynomial time and the fact that the matrix rank over any field can be determined in polynomial time (for example by the method of Gaussian elimination), this ensures that all the logics which we introduce in the following have polynomial-time data complexity.

Let $\bar{x}\bar{\nu}$ be a mixed tuple of variables and let $\bar{t}$ be a tuple of closed numeric terms which bounds the range of the numeric variables in $\bar{\nu}$. For a formula φ we define a *counting term* $s = [\#\bar{x}\bar{\nu} \leq \bar{t}\,.\,\varphi]$ whose value $s^{\mathfrak{A}} \in \mathbb{N}$ in a structure $\mathfrak{A}$ corresponds to the number of tuples $(\bar{a}, \bar{n}) \in A^k \times \mathbb{N}^\ell$ such that $\mathfrak{A} \models \varphi(\bar{a}, \bar{n})$ and $n_i \leq t_i^{\mathfrak{A}}$ where $k = |\bar{x}|$ and $\ell = |\bar{\nu}|$ (to be precise, we should write $\mathfrak{A}^{\#}$ instead of $\mathfrak{A}$, but we usually omit the superscript for the sake of better readability). We then define *first-order logic with counting* FOC as the extension of (the above described two-sorted variant of) FO by counting terms. Similarly, by adding counting terms to the logic FP we obtain *(inflationary) fixed-point logic with counting* FPC.

Rank operators. Let $\Theta(\bar{x}\bar{\nu}, \bar{y}\bar{\mu})$ be a numeric term and let $\bar{t}$ and $\bar{s}$ be tuples of closed numeric terms which bound the range of the numeric variables in $\bar{\nu}$ and $\bar{\mu}$, respectively. Given a structure $\mathfrak{A}$ we define $\mathbb{N}^{\leq \bar{t}} := \{\bar{n} \in \mathbb{N}^{|\bar{\nu}|} : n_i \leq t_i^{\mathfrak{A}}\}$. The set $\mathbb{N}^{\leq \bar{s}} \subset \mathbb{N}^{|\bar{\mu}|}$ is defined analogously. The term Θ together with $\bar{t}$ and $\bar{s}$ defines in the structure $\mathfrak{A}$ for $I := A^{|\bar{x}|} \times \mathbb{N}^{\leq \bar{t}}$ and $J := A^{|\bar{y}|} \times \mathbb{N}^{\leq \bar{s}}$ the $I \times J$-matrix M_Θ with values in $\mathbb{N}$ given as $M_\Theta(\bar{a}\bar{n}, \bar{b}\bar{m}) := \Theta^{\mathfrak{A}}(\bar{a}\bar{n}, \bar{b}\bar{m})$.

The *matrix rank operators* compute the rank of the matrix M_Θ over a prime field $\mathbb{F}_p$ for $p \in \mathbb{P}$. First, as in [7], we define for every prime p a matrix rank operator rk_p which allows us to construct a new numeric *rank term* $[\mathsf{rk}_p\,(\bar{x}\bar{\nu} \leq \bar{t}, \bar{y}\bar{\mu} \leq \bar{s})\,.\,\Theta]$ whose value in the structure $\mathfrak{A}$ is the rank of the matrix $(M_\Theta \bmod p)$ over $\mathbb{F}_p$. Secondly, we propose a uniform rank operator rk^* which takes the prime p as an additional input. Formally, with this rank operator rk^* we can construct a rank term $[\mathsf{rk}^*\,(\bar{x}\bar{\nu} \leq \bar{t}, \bar{y}\bar{\mu} \leq \bar{s}, \pi \leq r)\,.\,\Theta]$ where π is an additional free numeric variable whose range is bounded by some closed numeric term r. Given a structure $\mathfrak{A}$ and an assignment $\pi \mapsto p$ for some prime $p \leq r^{\mathfrak{A}}$, the value of the rank term is the matrix rank of $(M_\Theta \bmod p)$ considered as a matrix over $\mathbb{F}_p$ (if $\pi \mapsto n$ for $n \notin \mathbb{P}$, then the value is zero). The rank operator rk^* is a unification for the the family of separate rank operators $(\mathsf{rk}_p)_{p \in \mathbb{P}}$ and has been introduced in [14, 15, 17].

We define, for every set of primes $\Omega \subseteq \mathbb{P}$, the extension FOR_Ω of FOC and the extension FPR_Ω of FPC by matrix rank operators rk_p with $p \in \Omega$. For convenience, we let $\mathrm{FOR} = \mathrm{FOR}_\mathbb{P}$ and $\mathrm{FPR} = \mathrm{FPR}_\mathbb{P}$. Similarly, we denote by FPR^* the extension of FPC by the uniform rank operator rk^*. We remark, that rank operators can directly simulate counting terms. For example we have that $[\#x\,.\,\varphi(x)] = [\mathsf{rk}_p\,(x, y)\,.\,(x = y \wedge \varphi(x))]$. Hence, we could equivalently define the rank logics $\mathrm{FOR}_\Omega, \mathrm{FPR}_\Omega$ and FPR^* as the extensions of (the two-sorted variants of) FO and FP, respectively.

Solvability quantifiers. We next introduce extensions by quantifiers which directly express the solvability problem for linear equation systems over finite fields. Besides the applications in this paper, an additional advantage of such quantifiers is that they can be generalised for linear equation systems over more general classes of domains, like rings, for which no appropriate notion of matrix rank exists, cf. [6].

Let $\Omega \subseteq \mathbb{P}$ be a set of primes. Then the *solvability logic* FPS_Ω extends the syntax of FPC for every $p \in \Omega$ by the following formula creation rule for *solvability quantifiers* slv_p.

- Let $\varphi(\bar{x}\bar{\nu}, \bar{y}\bar{\mu}, \bar{z}) \in \mathrm{FPS}_\Omega$ and let $\bar{t}$ and $\bar{s}$ be tuples of closed numeric terms with $|\bar{t}\,| = |\bar{\nu}|$ and $|\bar{s}\,| = |\bar{\mu}|$. Then also $\psi(\bar{z}) = (\mathsf{slv}_p\,\bar{x}\bar{\nu} \leq \bar{s}, \bar{y}\bar{\mu} \leq \bar{t})\varphi(\bar{x}\bar{\nu}, \bar{y}\bar{\mu}, \bar{z})$ is a formula of FPS_Ω.

The semantics of the formula $\psi(\bar{z})$ is defined similarly as for rank logic. More precisely, let $k = |\bar{x}|$ and $\ell = |\bar{y}|$. To a pair $(\mathfrak{A}, \bar{z} \mapsto \bar{c}) \in \mathcal{S}(\sigma, \bar{z})$ we associate the $I \times J$-matrix M_φ over $\{0, 1\} \subseteq \mathbb{F}_p$ where $I = A^k \times \mathbb{N}^{\leq \bar{s}}$ and $J = A^\ell \times \mathbb{N}^{\leq \bar{t}}$ and where for $\bar{a} \in I$ and $\bar{b} \in J$ we have $M_\varphi(\bar{a}, \bar{b}) = 1$ if, and only if, $\mathfrak{A} \models \varphi(\bar{a}, \bar{b}, \bar{c})$.

Let $\mathbb{1}$ be the I-identity vector over $\mathbb{F}_p$, i.e. $\mathbb{1}(\bar{a}) = 1$ for all $\bar{a} \in I$. Then M_φ and $\mathbb{1}$ determine the linear equation system $M_\varphi \cdot \vec{x} = \mathbb{1}$ over $\mathbb{F}_p$ where $\vec{x} = (x_j)_{j \in J}$ is a J-vector of variables x_j which range over $\mathbb{F}_p$. Finally, $\mathfrak{A} \models \psi(\bar{c})$ if, and only if, $M_\varphi \cdot \vec{x} = \mathbb{1}$ is solvable.

At first glance, the solvability quantifier seem to pose serious restrictions on the syntactic form of definable linear equation systems. Specifically, the coefficient matrix has to be a matrix over $\{0, 1\}$ and the vector of constants is fixed from outside. However, it is not hard to show that general linear equation systems can be brought into this kind of normal form by using quantifier-free first-order transformations (see Lemma 4.1 in [6]).

We write FPS to denote the logic $\mathrm{FPS}_\mathbb{P}$ and FPS_p to denote the logic $\mathrm{FPS}_{\{p\}}$ for $p \in \mathbb{P}$. Analogously to the definition of FPR^* we also consider a solvability quantifier slv which

gets the prime p as an additional input and which can uniformly simulate all solvability quantifiers slv_p for $p \in \mathbb{P}$. Let FPS^* denote the extension of FPC by this uniform version of a solvability quantifier. The following inclusions follow from the definitions and the fact that rank operators can be used to define the solvability problem for linear equation systems.

$$\begin{array}{ccccccccc} \mathrm{FOR}_p & \leq & \mathrm{FPR}_p & \leq & \mathrm{FPR} & \leq & \mathrm{FPR}^* & \leq & \textsc{Ptime} \\ \wr & & \wr & & \wr & & \wr & & \\ \mathrm{FOS}_p & \leq & \mathrm{FPS}_p & \leq & \mathrm{FPS} & \leq & \mathrm{FPS}^* & & \\ \wr & & \wr & & & & & & \\ \mathrm{FO} & \leq & \mathrm{FPC} & & & & & & \end{array}$$

Finally we remark that, analogously to [7], we defined rank operators and solvability quantifiers for prime fields only. Of course, the definition can easily be generalised to cover all finite fields, i.e. also finite fields of prime power order. However, for the case of solvability quantifiers, Holm was able to prove in [14] that this does not alter the expressive power of the resulting logics since solvability quantifiers over a finite field $\mathbb{F}_q$ of prime power order $q = p^k$ can be simulated by solvability quantifiers over $\mathbb{F}_p$. In fact, a similar reduction can be achieved for rank operators which justifies to focus on rank operators and solvability quantifiers over prime fields.

3 Separation results over different classes of fields

In this section we separate the extensions FPS_Ω of fixed-point logic with counting by solvability quantifier for different sets of primes. Moreover, we transfer these results to the extensions FPR_Ω by rank operators.

▶ **Theorem 1.** *Let $\Omega \neq \Omega'$ be two sets of primes. Then $\mathrm{FPS}_\Omega \neq \mathrm{FPS}_{\Omega'}$ and $\mathrm{FPR}_\Omega \neq \mathrm{FPR}_{\Omega'}$.*

▶ **Theorem 2.** *Rank logic fails to capture polynomial time. We have $\mathrm{FPR} < \mathrm{FPR}^* \leq \textsc{Ptime}$.*

In fact, both theorems are simple consequences of our following main result.

▶ **Theorem 3.** *For every prime q there is a class of structures $\mathcal{K}_q$ such that*
(a) $\mathrm{FPS}_\Omega = \mathrm{FPC}$ *on $\mathcal{K}_q$ for every set of primes Ω with $q \notin \Omega$,*
(b) $\mathrm{FPR}_\Omega = \mathrm{FPS}_\Omega$ *on $\mathcal{K}_q$ for all sets of primes Ω,*
(c) $\mathrm{FPC} < \textsc{Ptime}$ *on $\mathcal{K}_q$, and*
(d) $\mathrm{FPS}_q = \textsc{Ptime}$ *on $\mathcal{K}_q$.*

Proof of Theorem 1. Without loss of generality let $q \in \Omega \setminus \Omega'$. Then by Theorem 3 there exists a class $\mathcal{K}_q$ on which $\mathrm{FPS}_\Omega = \mathrm{FPR}_\Omega = \textsc{Ptime}$ and $\mathrm{FPS}_{\Omega'} = \mathrm{FPR}_{\Omega'} = \mathrm{FPC} < \textsc{Ptime}$. ◀

Proof of Theorem 2. Assume that $\mathrm{FPR} = \textsc{Ptime}$. Then, in particular, $\mathrm{FPR} = \mathrm{FPR}^*$ and there exists a formula $\varphi \in \mathrm{FPR}$ which can uniformly determine the rank of matrices over prime fields, i.e. which can express the uniform rank operator rk^*. As a matter of fact we have $\varphi \in \mathrm{FPR}_\Omega$ for some *finite* set of primes Ω. By using φ we can uniformly express the matrix rank over each prime field $\mathbb{F}_p$ in FPR_Ω. In other words, we have $\mathrm{FPS} \leq \mathrm{FPR} \leq \mathrm{FPR}^* \leq \mathrm{FPR}_\Omega$.

Now let $q \in \mathbb{P} \setminus \Omega$. By Theorem 3 there exists a class of structures $\mathcal{K}_q$ on which $\mathrm{FPR}_\Omega = \mathrm{FPC} < \textsc{Ptime}$. However, the class $\mathcal{K}_q$ can be chosen such that $\textsc{Ptime} = \mathrm{FPS}_q \leq \mathrm{FPR}_\Omega$ on $\mathcal{K}_q$ by Theorem 3 (d) and we obtain the desired contradiction. ◀

The proof of Theorem 2 reveals a deficiency of the logic FPR: each formula can only access rk_p-operators for a finite set Ω of distinct primes p. In fact, the query which we constructed to separate FPR from PTIME can be defined in FPR^*. Altogether this suggests to generalise the notion of rank operators and to specify the prime p as a part of the input, as we did for FPR^*, and as it was proposed in [14, 15, 17].

The proof of Theorem 3 is structured as follows. We fix a prime q and identify, in a first step, sufficient criteria (i)–(iv) of classes of structures $\mathcal{K} = \mathcal{K}_q$ which guarantee that the relations claimed in (a), (b), (c) and (d) hold. In a second step, we construct a class of structures $\mathcal{K}$ and verify, in a third step, that $\mathcal{K}$ satisfies these sufficient criteria.

Establishing sufficient criteria. We start to find sufficient criteria for part (a) of Theorem 3.

(i) The automorphism groups $\Delta_{\mathfrak{A}} := \mathrm{Aut}(\mathfrak{A})$ of structures $\mathfrak{A} \in \mathcal{K}$ are Abelian q-groups.

(ii) The orbits of ℓ-tuples in structures $\mathfrak{A} \in \mathcal{K}$ can be ordered in FPC:
For all $\ell \geq 1$ there exists $\varphi_{\preceq}(x_1, \dots, x_\ell, y_1, \dots, y_\ell) \in \mathrm{FPC}$ such that for all $\mathfrak{A} \in \mathcal{K}$, the formula $\varphi_{\preceq}(\bar{x}, \bar{y})$ defines in $\mathfrak{A}$ a linear preorder $\preceq$ on A^ℓ with the property that two ℓ-tuples $\bar{a}, \bar{b} \in A^\ell$ are $\preceq$-equivalent if, and only if, they are in the same $\Delta_{\mathfrak{A}}$-orbit.

▶ **Lemma 4.** *If $\mathcal{K}$ satisfies* (i) *and* (ii)*, then* $\mathrm{FPS}_\Omega = \mathrm{FPC}$ *on $\mathcal{K}$ for all $\Omega \subseteq \mathbb{P} \setminus \{q\}$.*

The only interesting step of an inductive translation is the case of a solvability formula

$$\psi(\bar{z}) = (\mathsf{slv}_p\, \bar{x}\bar{\nu} \preceq \bar{s}, \bar{y}\bar{\mu} \preceq \bar{t})\varphi(\bar{x}\bar{\nu}, \bar{y}\bar{\mu}, \bar{z}).$$

Let $|\bar{x}| = |\bar{y}| = \ell$, $|\bar{\nu}| = |\bar{\mu}| = \lambda$ and $|\bar{z}| = k$. To explain our main argument, we fix a structure $\mathfrak{A} \in \mathcal{K}$ and a k-tuple of parameters $\bar{c} \in (A \uplus \mathbb{N})^k$ which is compatible with the type of $\bar{z}$. According to the semantics of the slv_p-quantifier, the formula φ defines in $(\mathfrak{A}, \bar{z} \mapsto \bar{c})$ an $I \times J$-matrix $M = M_{\bar{c}}^{\mathfrak{A}}$ over $\{0,1\} \subseteq \mathbb{F}_p$ where $I = I^{\mathfrak{A}} := A^\ell \times \mathbb{N}^{\preceq \bar{s}} \subseteq A^\ell \times \mathbb{N}^\lambda$ and $J = J^{\mathfrak{A}} := A^\ell \times \mathbb{N}^{\preceq \bar{t}} \subseteq A^\ell \times \mathbb{N}^\lambda$ that is defined for $\bar{a} \in I$ and $\bar{b} \in J$ as $M(\bar{a}, \bar{b}) = 1$ if, and only if, $\mathfrak{A} \vDash \varphi(\bar{a}, \bar{b}, \bar{c})$. Moreover, we have $\mathfrak{A} \vDash \psi(\bar{c})$ if, and only if, the linear system $M \cdot \vec{x} = \mathbb{1}$ over $\mathbb{F}_p$ is solvable. The key idea of our proof is to use the symmetries of the structure $\mathfrak{A}$ to translate the linear equation system $M \cdot \vec{x} = \mathbb{1}$ into an equivalent linear system for which the solvability problem is FPC-definable.

We set $\Gamma = \Gamma_{\bar{c}}^{\mathfrak{A}} := \mathrm{Aut}(\mathfrak{A}, \bar{c}) \leq \Delta = \Delta_{\mathfrak{A}} = \mathrm{Aut}(\mathfrak{A})$. The group Γ acts on I and J in the natural way. We identify each automorphism $\pi \in \Gamma$ with the corresponding $I \times I$-permutation matrix Π_I and the corresponding $J \times J$-permutation matrix Π_J in the usual way. More precisely, to $\pi \in \Gamma$ we associate the $I \times I$-permutation matrix Π_I with entries $\{0,1\}$ which is defined as $\Pi_I(\bar{a}, \bar{b}) = 1$ if, and only if, $\pi(\bar{a}) = \bar{b}$. Then Γ acts on the set of $I \times J$-matrices by left multiplication with $I \times I$-permutation matrices. Analogously, we let Π_J denote the $J \times J$-permutation matrix with entries $\{0,1\}$ that is defined in the same way as Π_I. Then Γ also acts on the set of $I \times J$-matrices by right multiplication with $J \times J$-permutation matrices. Specifically, for $\pi \in \Gamma$ we have $(\Pi_I \cdot M)(\bar{a}, \bar{b}) = M(\pi(\bar{a}), \bar{b})$ and $(M \cdot \Pi_J^{-1})(\bar{a}, \bar{b}) = M(\bar{a}, \pi(\bar{b}))$. Since M is defined by a formula in the structure $(\mathfrak{A}, \bar{c})$ and since $\Gamma = \mathrm{Aut}(\mathfrak{A}, \bar{c})$ we conclude that $(\Pi_I \cdot M \cdot \Pi_J^{-1})(\bar{a}, \bar{b}) = M(\pi(\bar{a}), \pi(\bar{b})) = M(\bar{a}, \bar{b})$ and thus

$$\Pi_I \cdot M \cdot \Pi_J^{-1} = M \quad \Leftrightarrow \quad \Pi_I \cdot M = M \cdot \Pi_J.$$

This identity leads to the following important observation.

▶ **Lemma 5.** *If $M \cdot \vec{x} = \mathbb{1}$ is solvable, then the system has a* Γ-symmetric *solution, i.e. a solution $\vec{b} \in \mathbb{F}_p^J$ such that $\Pi_J \cdot \vec{b} = \vec{b}$ for all $\pi \in \Gamma$.*

Proof. If $M \cdot \vec{b} = \mathbb{1}$, then also $\Pi_I \cdot (M \cdot \vec{b}) = \mathbb{1}$ and thus $M \cdot (\Pi_J \cdot \vec{b}) = \mathbb{1}$ for all $\pi \in \Gamma$. This shows that Γ acts on the solution space of the linear equation system. Since $\mathcal{K}$ satisfies property (i) we know that Γ is a q-group for a prime $q \neq p$. Thus each Γ-orbit has size q^r for some $r \geq 0$. On the other hand, the number of solutions is a power of p. We conclude that there is at least one Γ-orbit which contains a single solution only. ◀

Let $\vec{b} \in \mathbb{F}_p^J$ be a Γ-symmetric solution. Then the entries of the solution $\vec{b}$ on Γ-orbits are constant: for $j \in J$ and $\pi \in \Gamma$ we have $\vec{b}(\pi(j)) = (\Pi_J \cdot \vec{b})(j) = \vec{b}(j)$. We use property (ii) to show that there is an FPC-formula $\varphi_\preceq(\bar{x}, \bar{y})$ which defines for all $\mathfrak{A} \in \mathcal{K}$ and $\bar{c} \in (A \uplus \mathbb{N})^k$ as above a linear preorder $\preceq$ on A^ℓ which identifies Γ-orbits. Note that, in general, $\Gamma = \mathrm{Aut}(\mathfrak{A}, \bar{c})$ is a strict subgroup of $\Delta = \mathrm{Aut}(\mathfrak{A})$. Thus we can not directly apply (ii). However, the Γ-orbits on A^ℓ correspond to the Δ-orbits on $A^{k'+\ell}$ where the first k' entries are fixed to the elements in $\{c_1, \dots, c_k\} \cap A$.

The linear preorder $\preceq$ naturally extends to a preorder on the sets I and J with the same properties. Let us write $J = J_0 \preceq J_1 \preceq \cdots \preceq J_{v-1}$ to denote the decomposition of J into the Γ-orbits J_j which are ordered by $\preceq$ as indicated. Moreover, for $j \in [v]$ we let e_j denote the identity vector on the j-th orbit J_j, i.e. the J-vector which defined for $i \in J$ as $e_j(i) = 1$ if $i \in J_j$ and as $e_j(i) = 0$ otherwise. Let E denote the $J \times [v]$-matrix whose j-th column is the vector e_j. It follows that a Γ-symmetric solution $\vec{b}$ can be written as $E \cdot \vec{b}_* = \vec{b}$ for a unique $[v]$-vector $\vec{b}_*$. Together with Lemma 5 this shows the following.

▶ **Lemma 6.** *The system $M \cdot \vec{x} = \mathbb{1}$ is solvable if, and only if, $(M \cdot E) \cdot \vec{x}_* = \mathbb{1}$ is solvable.*

Finally, we observe that the coefficient matrix $M_* := (M \cdot E)$ of the equivalent linear equation system $M_* \cdot \vec{x}_* = \mathbb{1}$ can easily be obtained in FPC and that it is a matrix over the *ordered* set of column indices $[v]$. It is a simple observation that such linear equation systems can be solved in FPC: the linear order on the column set induces (together with some fixed order on $\mathbb{F}_p$) a lexicographical ordering on the set of rows which is, up to duplicates of rows, a linear order on this set. Thus, in general, if we have a linear order on *one* of the index sets of the coefficient matrix this suffices to obtain an equivalent matrix where *both* index sets are ordered, see also [17]. This finishes our proof of Lemma 4.

We proceed to show that the conditions (i) and (ii) also guarantee that rank operators can be reduced to solvability operators over the class

▶ **Lemma 7.** *If $\mathcal{K}$ satisfies* (i) *and* (ii)*, then $\mathrm{FPR}_\Omega = \mathrm{FPS}_\Omega$ on $\mathcal{K}$ for all sets of primes Ω.*

Proof. The only interesting case of an inductive translation is the case of rank terms

$$\Upsilon(\bar{z}) = [\mathsf{rk}_p\,(\bar{x}\bar{\nu} \preceq \bar{t}, \bar{y}\bar{\mu} \preceq \bar{s})\,.\,\Theta(\bar{x}\bar{\nu}, \bar{y}\bar{\mu}, \bar{z})].$$

Let $|\bar{x}| = |\bar{y}| = \ell$, $|\bar{\nu}| = |\bar{\mu}| = \lambda$ and $|\bar{z}| = k$. Let $\mathfrak{A} \in \mathcal{K}$ and let $\bar{c}$ be a k-tuple of parameters $\bar{c} \in (A \uplus \mathbb{N})^k$ which is compatible with $\bar{z}$. The term Θ defines in $(\mathfrak{A}, \bar{z} \mapsto \bar{c})$ for $I^\mathfrak{A} = I := A^{|\bar{x}|} \times \mathbb{N}^{\leq \bar{t}}$ and $J^\mathfrak{A} = J := A^{|\bar{y}|} \times \mathbb{N}^{\leq \bar{s}}$ the $I \times J$-matrix M over $\mathbb{F}_p$ which is defined as $M(\bar{a}\bar{n}, \bar{b}\bar{m}) := \Theta^\mathfrak{A}(\bar{a}\bar{n}, \bar{b}\bar{m}, \bar{c}) \bmod p$.

We proceed to show that we can obtain the matrix rank of M, that is the value $\Upsilon^\mathfrak{A}(\bar{c}) \in \mathbb{N}$, by a recursive application of solvability queries. We first make the following key observation.

Claim: There are FPC-formulas $\varphi_\preceq(\bar{y}_1\bar{\mu}_1, \bar{y}_2\bar{\mu}_2)$, $\psi_\preceq(\bar{v}, \bar{y}_1\bar{\mu}_1, \bar{y}_2\bar{\mu}_2)$ such that for every $\mathfrak{A} \in \mathcal{K}$

(a) $\varphi_\preceq^\mathfrak{A}$ is a linear preorder $\preceq$ on $J^\mathfrak{A}$, and such that

(b) for every $\preceq$-class $[j] \subseteq J^\mathfrak{A}$ there exists $\bar{d} \in A^{|\bar{v}|}$ such that $\psi_\preceq^\mathfrak{A}(\bar{d})$ is a linear order on $[j]$.

Proof of claim: We use property (ii) to choose an FPC-formula $\varphi_\preceq$ which defines in all $\mathfrak{A} \in \mathcal{K}$ a linear preorder $\preceq$ on $J^{\mathfrak{A}}$ such that $\preceq$-classes correspond to $\Delta_{\mathfrak{A}}$-orbits Analogously, we choose an FPC-formula $\vartheta_\preceq$ which defines in every structure $\mathfrak{A} \in \mathcal{K}$ a linear preorder $\preceq^*$ on $J^{\mathfrak{A}} \times J^{\mathfrak{A}}$ and that induces a linear order on the $\Delta_{\mathfrak{A}}$-orbits.

To obtain $\psi_\preceq$, we let $[j] \subseteq J^{\mathfrak{A}}$ be a $\preceq$-class for some $\mathfrak{A} \in \mathcal{K}$. By property (i) we know that $\Delta_{\mathfrak{A}}$ is an Abelian group. Thus, each automorphism $\pi \in \Delta_{\mathfrak{A}}$ which fixes *one* element in the $\Delta_{\mathfrak{A}}$-orbit $[j]$ point-wise fixes *every* element in the class $[j]$. We conclude that the restriction of $\preceq^*$ to elements in $\{j'\} \times [j]$ corresponds to a linear order on $[j]$ for each $j' \in [j]$. ⊣

We are prepared to describe the recursive procedure which allows us to determine the rank of the matrix M in FPS_Ω. We fix formulas $\varphi_\preceq$ and $\psi_\preceq$ with the properties stated in the claim above. Moreover, let $\preceq$ denote the linear preorder defined by $\varphi_\preceq$ on $J = J_0 \preceq J_1 \preceq \cdots \preceq J_{r-1}$. We use the formula $\psi_\preceq$ to obtain on each class J_i a family of definable linear orderings (which depend on the choice of different parameters). For $j \in J$ we denote by $\vec{m}_j \in \mathbb{F}_p^I$ the j-th column of the matrix M. Then the rank of M coincides with the dimension of the $\mathbb{F}_p$-vector space which is generated by the set of columns $\{\vec{m}_j : j \in J\}$ of the matrix M.

Now, for $i \in [r]$ we recursively obtain the dimension $d_i \in \mathbb{N}$ of the $\mathbb{F}_p$-vector space generated by $V_i := \{\vec{m}_j : j \in J_0 \cup J_1 \cup \cdots \cup J_i\}$ as follows. First, we use $\psi_\preceq$ to fix a linear order on J_i (the following steps are independent of the specific linear order and can thus be performed in parallel for each such order). Using this linear order on J_i we can identify in FPS_Ω a maximal set $W \subseteq \{\vec{m}_j : j \in J_i\}$ of linearly independent columns such that $\langle V_{i-1}\rangle \cap \langle W\rangle = \{\vec{0}\}$. Indeed, if $\langle V_{i-1}\rangle \cap \langle W\rangle = \{\vec{0}\}$, then for $\vec{m} \in \{\vec{m}_j : j \in J_i\}$, $\vec{m} \notin \langle W\rangle$ we have that $\langle V_{i-1}\rangle \cap \langle W \uplus \{\vec{m}\}\rangle = \{\vec{0}\}$ if, and only if, $\vec{m} \notin \langle V_{i-1} \cup W\rangle$. Observe that the conditions $\vec{m} \notin \langle W\rangle$ and $\vec{m} \notin \langle V_{i-1} \cup W\rangle$ correspond to the solvability of a linear equation system over $\mathbb{F}_p$. We claim that $d_i = d_{i-1} + |W|$. Indeed, by the maximality of W and since $\langle V_{i-1}\rangle \cap \langle W\rangle = \{\vec{0}\}$ it follows that $\langle V_i\rangle = \langle V_{i-1}\rangle \oplus \langle W\rangle$. Moreover, W consists of linearly independent columns and is a basis for $\langle W\rangle$.

Since the above described recursion can easily be implemented in FPS_Ω, we conclude that the rank d_{r-1} of the matrix M can be determined in FPS_Ω which completes our proof. ◀

We now focus on the parts (c) and (d) of Theorem 3.

(iii) There exists an FPS_q-definable canonisation procedure on $\mathcal{K}$.

(iv) For all $k \geq 1$ there is a pair $\mathfrak{A}, \mathfrak{B} \in \mathcal{K}$ such that $\mathfrak{A} \not\cong \mathfrak{B}$ and $\mathfrak{A} \equiv_k^C \mathfrak{B}$ (that is, $\mathfrak{A}$ and $\mathfrak{B}$ cannot be distinguished in the k-variable fragment of infinitary counting logic $\mathrm{C}^k_{\infty\omega}$).

▶ **Lemma 8.** *If $\mathcal{K}$ satisfies* (iii) *and* (iv)*, then* $\mathrm{FPC} < \mathrm{FPS}_q = \textsc{Ptime}$ *on* $\mathcal{K}$.

Constructing an appropriate class of structures. We proceed to construct a class of structures $\mathcal{K}$ which satisfies properties (i)–(iv). Our approach is a generalisation of the well-known construction of Cai, Fürer and Immerman [4] for fields $\mathbb{F}_q$, $q \in \mathbb{P}$. The difference to the original construction (which arises as a special case for $q = 2$) is that we replace every edge e from the original graph $\mathcal{G}$ by q copies $e_0, e_1, \ldots, e_{q-1}$ which we arrange on a directed cycle of length q. For $q = 2$ this is equivalent to just taking two non-connected atoms e_0, e_1. While the symmetries of the original CFI-graphs arise by twisting pairs of corresponding edges e_0, e_1, the symmetries of generalised CFI-structures arise by shifting the cycles on $e_0, e_1, \ldots, e_{q-1}$ by some value $x \in \mathbb{F}_q$. In both cases, the resulting twists and cyclic shifts can be propagated along paths in $\mathcal{G}$. We remark that the same kind of generalisations have been studied, for example, in [14, 18]. Due to space limitations, we have to leave out the following proofs which are mostly straightforward adaptations of the arguments for the original construction.

We start from an *(undirected), connected* and *ordered* graph $\mathcal{G} = (V, \leq, E)$. Let C, I and R be binary relation symbols. We set $\tau := \{\leq, C, I, R\}$. We define for every prime q and every sequence of *gadget values* $\vec{d} = (d_v)_{v \in V} \in [q]^V$ a τ-structure $\mathrm{CFI}_q(\mathcal{G}, \vec{d})$ which we call a *CFI-structure over* $\mathcal{G}$. For the following construction we agree that arithmetic is modulo q so that we can drop the operator "mod q" in statements of the form $x = y \bmod q$ and $x + y \bmod q$ for the sake of better readability. For what follows, let $E(v) \subseteq E$ denote the set of *directed* edges starting in v. Since $\mathcal{G}$ is an undirected graph, this means that for each undirected edge $\{v, w\}$ of $\mathcal{G}$ we have $(v, w) \in E(v)$ and $(w, v) \in E(w)$.

- The *universe* of $\mathrm{CFI}_q(\mathcal{G}, \vec{d})$ consists of *edge nodes* and *equation nodes*.
  - The set of *edge nodes* $\hat{E}$ is defined as $\hat{E} := \bigcup_{e \in E} \hat{e}$ where for every *directed* edge $e \in E$ we let the *edge class* $\hat{e} = \{e_0, e_1, \ldots, e_{q-1}\}$ consist of q distinct copys of e. In particular, for every edge $e = (v, w) \in E$ and its reversed edge $e^{-1} := f = (w, v) \in E$ the sets $\hat{e}$ and $\hat{f}$ are disjoint. We say that two such edges (or edge classes) are *related*.
  - The set of *equation nodes* $\hat{V}$ is defined as $\hat{V} := \bigcup_{v \in V} \hat{v}^{\vec{d}(v)}$ where for every vertex $v \in V$ and $d \in [q]$ the *equation class* $\hat{v}^d$ consist of all functions $\rho : E(v) \to [q]$ which satisfy $\sum \rho := \sum_{e \in E(v)} \rho(e) = d$.
- The *linear preorder* $\preceq$ orders the edge classes according to the linear order induced by $\leq$ on E. More precisely, we let $\hat{e} \preceq \hat{f}$ whenever $e \leq f$. Similarly, $\preceq$ orders the equation classes according to the order of $\leq$ on V, i.e. $\hat{v} \preceq \hat{w}$ if $v \leq w$. Moreover, we let $\hat{e} \preceq \hat{v}$ for edge classes $\hat{e}$ and equation classes $\hat{v}$.
- The *cycle relation* C contains a directed cycle of length q on each of the edge classes $\hat{e}$ for $e \in E$, i.e. $C = \{(e_i, e_{i+1}) : i \in [q], e \in E\}$.
- The *inverse relation* I connects two related edge classes by pairing additive inverses. More precisely, let $e = (v, w) \in E$ and $f = (w, v) \in E$. Then I contains all edges (e_x, f_y) with $x + y = 0$ for $x, y \in [q]$.
- The *gadget relation* R is defined as $R := \bigcup_{v \in V} R_v^{\vec{d}(v)}$ where for $v \in V$ and $d \in [q]$ the relation R_v^d is given as

$$R_v^d := \{(\rho, e_{\rho(e)}) : \rho \in \hat{v}^d, e \in E(v)\}.$$

At first glance our construction associates to every graph $\mathcal{G}$ (with the above properties) and to each sequence of gadget values $\vec{d} \in [q]^V$ a different structure $\mathrm{CFI}_q(\mathcal{G}, \vec{d})$. However, for each graph $\mathcal{G}$ with the above properties there really are, up to isomorphism, only q different CFI-structures $\mathrm{CFI}_q(\mathcal{G}, \vec{d})$.

▶ **Lemma 9.** *Let* $\vec{d}, \vec{d}_* \in ([q])^V$. *Then* $\mathrm{CFI}_q(\mathcal{G}, \vec{d}) \cong \mathrm{CFI}_q(\mathcal{G}, \vec{d}_*)$ *if, and only if,* $\sum \vec{d} = \sum \vec{d}_*$.

A connected graph $\mathcal{G}$ is *k-connected*, for $k \geq 0$, if $\mathcal{G}$ contains more than k vertices and if $\mathcal{G}$ stays connected when we remove any set of at most k vertices. The *connectivity* $\mathrm{con}(\mathcal{G})$ of $\mathcal{G}$ is the maximal $k \geq 0$ such that $\mathcal{G}$ is k-connected. Moreover, the *connectivity* $\mathrm{con}(\mathfrak{G})$ of a class $\mathfrak{G}$ of connected graphs is the function $\mathrm{con}(\mathfrak{G}) : \mathbb{N} \to \mathbb{N}$ defined as

$$n \mapsto \min_{\mathcal{G} \in \mathfrak{G}, |\mathcal{G}| = n} \mathrm{con}(\mathcal{G}).$$

We are prepared to define the class $\mathcal{K}$: let $\mathfrak{G}$ be a class of *undirected, ordered, connected* graphs such that $\mathrm{con}(\mathfrak{G}) \in \omega(1)$ (for example complete, ordered graphs). Then we set

$$\mathcal{K} = \mathcal{K}_q := \{\mathrm{CFI}_q(\mathcal{G}, \vec{d}) : \mathcal{G} = (V, \leq, E) \in \mathfrak{G}, \vec{d} \in [q]^V\}.$$

Verifying the required properties. First of all, one can see that the cycle relation C and the preorder $\preceq$ enforce that the automorphism group of a CFI-structure $\mathrm{CFI}_q(\mathcal{G}, \vec{d})$ over $\mathcal{G} = (V, \leq, E) \in \mathfrak{G}$ is a subgroup of $\mathbb{F}_q^E$. Thus property (i) holds for $\mathcal{K}$.

To show that $\mathcal{K}$ satisfies property (ii), we fix the length $\ell \geq 1$ of tuples on which we want to define a linear preorder which identifies $\Delta_{\mathfrak{A}}$-orbits. By the definition of $\mathcal{K}$ it suffices to consider CFI-structures $\mathfrak{A} = \mathrm{CFI}_q(\mathcal{G}, \vec{d})$ over graphs $\mathcal{G} = (V, \leq, E) \in \mathfrak{G}$ with $\mathrm{con}(\mathcal{G}) > (\ell + 2)$ since almost all structures in $\mathcal{K}$ satisfy this condition. Then we can show that the equivalence classes of ℓ-tuples in the infinitary logic with counting and $(\ell + 2)$ variables $\mathrm{C}^{\ell+2}_{\infty\omega}$ coincides with the $\Delta_{\mathfrak{A}}$-orbits of ℓ-tuples for structures $\mathfrak{A} \in \mathcal{K}$.

▶ **Lemma 10.** *Let $\lambda \leq \ell$ and let $\bar{a}, \bar{b} \in A^\lambda$. Then $(\mathfrak{A}, \bar{a}) \equiv^C_{\ell+2} (\mathfrak{A}, \bar{b})$ if, and only if, there exists $\pi \in Aut(\mathfrak{A})$ such that $\pi(\bar{a}) = \bar{b}$.*

It is well-known that classes of $\mathrm{C}^{\ell+2}_{\infty\omega}$-equivalent tuples can be ordered in FPC, see e.g. [16]. Hence, it follows from our previous lemma that the class $\mathcal{K}$ satisfies property (ii).

▶ **Lemma 11.** *The class $\mathcal{K}$ satisfies the properties* (i) *and* (ii).

We turn our attention to property (iv). In the next lemma we state that for each $k \geq 1$ and each sufficiently connected graph $\mathcal{G} \in \mathfrak{G}$, the logic $\mathrm{C}^k_{\infty\omega}$ cannot distinguish between any pair of CFI-structures over $\mathcal{G}$ (although there exist non-isomorphic CFI-structures over $\mathcal{G}$).

▶ **Lemma 12.** *Let $k \geq 1$ and let $\mathcal{G} = (V, \leq, E) \in \mathfrak{G}$ such that $con(\mathcal{G}) > k$. Then for all $\vec{d}, \vec{d}_* \in [q]^V$ it holds that $\mathrm{CFI}_q(\mathcal{G}, \vec{d}) \equiv^C_k \mathrm{CFI}_q(\mathcal{G}, \vec{d}_*)$. Thus, $\mathcal{K}$ satisfies property (iv).*

To complete our proof we establish an FPS_q-definable canonisation procedure on $\mathcal{K}$. The idea is as follows: given a CFI-structure $\mathfrak{A} = \mathrm{CFI}_q(\mathcal{G}, \vec{d})$ and a value $z \in [q]$ we construct a linear equation system over $\mathbb{F}_q$ which is solvable if, and only if, $\sum \vec{d} = z$. This linear equation system is FO-definable in $\mathfrak{A}$ which shows that FPS_q can determine the isomorphism class of a CFI-structure over $\mathcal{G}$. Since the graph $\mathcal{G}$ is ordered it is easy to construct an ordered representative from each isomorphism class of CFI-structures over $\mathcal{G}$.

More specifically, let $\mathcal{G} = (V, \leq, E) \in \mathfrak{G}$, let $\mathfrak{A} = \mathrm{CFI}_q(\mathcal{G}, \vec{d}) \in \mathcal{K}$ and let $z \in \mathbb{F}_q$. For our linear equation system we identify each element $e_i \in \hat{E}$ and each vertex $v \in V$ with a variable over $\mathbb{F}_q$, i.e. we let $\mathcal{V} := \hat{E} \uplus V$ be the set of variables. The equations are given as follows:

$$e_{i+1} = e_i + 1 \qquad \text{for all } e_i \in \hat{E} \tag{E 1}$$

$$e_i = -f_{-i} \qquad \text{for related edges } e, f \in E \tag{E 2}$$

$$v = \sum_{e \in E(v)} e_{\rho(e)} \qquad \text{for all } v \in V, \rho \in \hat{v} \tag{E 3}$$

$$z = \sum_{v \in V} v. \tag{E 4}$$

It is easy to see that this system is FO-definable in $\mathfrak{A}$. First of all, the equation (E 4) can be defined as a sum over the ordered set V. Moreover, we can express the equations of type (E 1) and (E 2) by using the cycle and inverse relation, respectively. Finally, the equations of type (E 3) can be expressed by using the gadget relation R.

▶ **Lemma 13.** *The above defined system is solvable if, and only if, $\sum \vec{d} = z$.*

▶ **Lemma 14.** *The class $\mathcal{K}$ satisfies the property* (iii).

4 Solvability quantifiers vs. rank operators

In the previous section we obtained separation results for the extensions of FPC by solvability quantifiers (and rank operators) over different sets of primes. One important step of our proof was to construct a class of structures on which the expressive power of the logics FPR_Ω and FPS_Ω coincides. Moreover, as we already mentioned in Section 2, most of the queries which are known to separate fixed-point logic with counting and rank logic can also be expressed in FPS. This naturally leads to the question whether, in general, rank operators can be simulated by solvability quantifiers in fixed-point logic with counting.

In this section we solve a simplified version of this question and show that in the absence of counting, rank operators are strictly more expressive than solvability quantifiers. The reader should recall that rank operators can easily simulate counting terms but this does not hold for solvability quantifiers.

To state our main result formally, we define for every prime p the extension FOS_p of first-order logic (without counting) by solvability quantifiers over $\mathbb{F}_p$. The crucial difference to the extension FOR_p of first-order logic by rank operators rk_p is that FOS_p is a *one-sorted* logic which does not have access to a counting sort.

▶ **Definition 15.** For every prime p, the logic FOS_p results by extending the syntax of FO by the following formula creation rule:

- If $\varphi(\bar{x},\bar{y},\bar{z}) \in \mathrm{FOS}_p$, then $\psi(\bar{z}) = (\mathsf{slv}_p\,\bar{x},\bar{y})\varphi(\bar{x},\bar{y},\bar{z})$ is an FOS_p-formula.

The semantics of $\psi(\bar{z})$ are defined as for FPS_p.

We briefly summarise what is known about FOS_p (see also [6, 17]). It follows from [7, 14] that for every prime p, the logic FOS_p subsumes the logic STC and that $\mathrm{FOS}_p \not\leq \mathrm{FPC}$. Moreover, on ordered structures, the expressive power of FOS_p can be characterised in terms of a natural complexity class: in [3], Buntrock et. al. introduced the *logarithmic space modulo counting classes* $\mathrm{MOD}_k\mathrm{L}$ for integers $k \geq 2$. Informally, a problem is in $\mathrm{MOD}_k\mathrm{L}$ if there exists a NL-Turing machine which verifies its inputs by producing a number of accepting paths which is not congruent 0 mod k. It turns out that, at least for primes p, the class $\mathrm{MOD}_p\mathrm{L}$ is closed under many natural operations, including all Boolean operations and even logspace Turing reductions [3]. Furthermore, many problems from linear algebra over $\mathbb{F}_p$ are complete for $\mathrm{MOD}_p\mathrm{L}$. In particular this is true for the solvability problem of linear equation systems over $\mathbb{F}_p$ and for computing the matrix rank over $\mathbb{F}_p$ [3].

Building on these insights, Dawar et. al. were able to show in [7] that for all $p \in \mathbb{P}$, the logic FOR_p captures $\mathrm{MOD}_p\mathrm{L}$ on the class of ordered structures. It has been noted in [17] that their proof shows the same correspondence for FOS_p.

▶ **Proposition 16** ([7],[17]). *On ordered structures we have* $\mathrm{FOS}_p = \mathrm{FOR}_p = \mathrm{MOD}_p\mathrm{L}$.

Despite this nice characterisation over ordered structures, the situation over general structures remained unclear. It easily follows that $\mathrm{FOS}_p \leq \mathrm{FOR}_p \leq \mathrm{FPR}_p$, but, so far, it has been open whether one, or both, of these inclusions are strict. In this section we show:

▶ **Theorem 17.** *For all primes p we have* $\mathrm{FOS}_p < \mathrm{FOR}_p$ *over the class of sets* $\mathcal{S}(\varnothing)$.

In some sense, this result is not very surprising. While FOS_p has to express $\mathcal{S}(\varnothing)$-properties over *unordered* sets, which have the maximal amount of symmetries, FOR_p can use the size of a set as a complete invariant to express properties of $\mathcal{S}(\varnothing)$-structures over the *ordered* numerical sort. However, it is not obvious how one can turn this intuition into a formal argument. In fact, FOS_p has non-trivial expressive power over sets. For instance,

FOS_p can determine the size of sets modulo p^k for every fixed k, while fixed-point logic FP, for example, collapses to first-order logic over sets.

To prove Theorem 17 we recall the following normal form for FOS_p which has been established in Corollary 4.8 of [6].

▶ **Theorem 18.** *Every formula $\vartheta(\bar{z}) \in \mathrm{FOS}_p$ is equivalent to an FOS_p-formula of the form $(\mathsf{slv}_p \bar{x}_1, \bar{x}_2)\alpha(\bar{x}_1, \bar{x}_2, \bar{z})$ where $\alpha(\bar{x}_1, \bar{x}_2, \bar{z})$ is quantifier-free.*

Similar to our approach in Section 3, the main idea for separating FOS_p and FOR_p is to exploit the symmetries of definable linear equation systems. More precisely, our plan is to considerably reduce the size of a given linear equation system along an FOR_p-definable transformation. For the remainder of this section, let us fix a quantifier-free formula $\alpha(x_1, \dots, x_k, y_1, \dots, y_\ell) \in \mathrm{FO}(\varnothing)$ and a prime p. According to the semantics of FOS_p, the formula α defines in an input structure $\mathfrak{A} = ([n])$ of size n the $[n]^k \times [n]^\ell$-coefficient matrix M_n which is given for $\bar{a} \in [n]^k, \bar{b} \in [n]^\ell$ as

$$M_n(\bar{a}, \bar{b}) = \begin{cases} 1, & \text{if } \mathfrak{A} \vDash \alpha(\bar{a}, \bar{b}) \\ 0, & \text{otherwise.} \end{cases}$$

Then $\mathfrak{A} \vDash (\mathsf{slv}_p \bar{x}_1, \bar{x}_2)\alpha(\bar{x}_1, \bar{x}_2)$ if the linear equation system $M_n \cdot \vec{x} = \mathbb{1}$ over $\mathbb{F}_p$ is solvable. For convenience we set $I_n = [n]^k$ and $J_n = [n]^\ell$.

Let $\Gamma = \Gamma_n = \mathrm{Sym}([n])$. Then the group Γ acts on I_n and J_n and we identify the action of $\pi \in \Gamma$ with the multiplication by the associated $I_n \times I_n$-permutation matrix Π_I and the $J_n \times J_n$-permutation matrix Π_J, respectively, as in Section 3. Hence, for $\pi \in \Gamma$ we have

$$\Pi_I \cdot M_n \cdot \Pi_J^{-1} = M_n \quad \Leftrightarrow \quad \Pi_I \cdot M_n = M_n \cdot \Pi_J.$$

For what follows, we fix a prime $q \neq p$ and a subgroup $\Delta \leq \Gamma$ such that $|\Delta| = q^m$ for some $m \geq 0$. The overall strategy is to use the Δ-symmetries of the matrix M_n to strongly reduce the size of the linear equation system $M_n \cdot \vec{x} = \mathbb{1}$. More precisely we claim that for $M_n^* := \sum_{\pi \in \Delta} \Pi_I \cdot M_n$ the linear equation system $M_n \cdot \vec{x} = \mathbb{1}$ is solvable if, and only if, $M_n^* \cdot \vec{x} = \mathbb{1}$ is solvable. First of all we note that for all $\pi \in \Delta$ we have:

- $\Pi_I \cdot M_n^* = \sum_{\lambda \in \Delta} \Pi_I \cdot \Lambda_I \cdot M_n = \sum_{\pi \in \Delta} \Pi_I \cdot M_n = M_n^*$
- $M_n^* \cdot \Pi_J = \sum_{\lambda \in \Delta} \Lambda_I \cdot M_n \cdot \Pi_J = \sum_{\lambda \in \Delta} \Lambda_I \cdot \Pi_I \cdot M_n = M_n^*$.

To verify our original claim assume that $M_n^* \cdot \vec{b} = \mathbb{1}$. Then we have

$$\mathbb{1} = M_n^* \cdot \vec{b} = (\sum_{\pi \in \Delta} \Pi_I \cdot M_n) \cdot \vec{b} = (\sum_{\pi \in \Delta} M_n \cdot \Pi_J) \cdot \vec{b} = M_n \cdot \sum_{\pi \in \Delta} (\Pi_J \cdot \vec{b}).$$

For the other direction let $M_n \cdot \vec{b} = \mathbb{1}$. Then $\sum_{\pi \in \Delta} \Pi_I \cdot M_n \cdot \vec{b} = |\Delta| \cdot \mathbb{1}$, hence $(1/|\Delta|) \cdot \vec{b}$ is a solution of the linear equation system $M_n^* \cdot \vec{x} = \mathbb{1}$. Note that for this direction we require that q and p are co-prime as we have to divide by $|\Delta|$.

Since M_n^* satisfies $\Pi_I \cdot M_n^* = M_n^* \cdot \Pi_J = M_n^*$ for all $\pi \in \Delta$ we have

$$M_n^*(\bar{a}, \bar{b}) = M_n^*(\pi(\bar{a}), \bar{b}) = M_n^*(\bar{a}, \pi(\bar{b}))$$

for all $\bar{a} \in I_n, \bar{b} \in J_n$ and $\pi \in \Delta$. In other words, the entries of the $I_n \times J_n$-matrix M_n^* are constant on the Δ-orbits of the index sets I_n and J_n. More specifically, if we let I_n^Δ and J_n^Δ denote the sets of Δ-orbits on I_n and J_n, respectively, then M_n^* can be identified with the matrix (M_n^*/Δ) which is defined as

$$(M_n^*/\Delta) : I_n^\Delta \times J_n^\Delta \to \mathbb{F}_p, ([\bar{a}], [\bar{b}]) \mapsto M_n^*(\bar{a}, \bar{b}).$$

Note that, depending on the size of the group Δ, the sets I_n^Δ and J_n^Δ can be noticeably smaller than the index sets I_n and J_n. Hence our obvious strategy is to choose Δ as large as possible to obtain a compact linear equation system $M_n^* \cdot \vec{x} = \mathbb{1}$ which is equivalent to the given one. It can be shown that for the case $n = q^r$, the size of the maximal q-subgroups Δ_n of Γ_n (the *q-Sylow subgroups*) is exponential in n and that the Δ_n-orbits on I_n and J_n can be described by a tuple of constant length with entries in $[r]$. Moreover, given a set $\mathfrak{A} = ([r])$ it is possible to construct in FOR_p the matrix $M_n^* = (M_n^*/\Delta_n)$ for $n = q^r$. In other words, FOR_p can equivalently express the solvability problem $M_n \cdot \vec{x} = \mathbb{1}$ defined by α in a structure of size $n = q^r$ in an exponentially more succinct structure of size r. The following lemma summarises this fact.

▶ **Lemma 19.** *There exists an* FOC*-term* $\Theta(\bar{\mu}, \bar{\nu})$ *which defines for all* $r \geq q$ *in the structure* $\mathfrak{A} = ([r])$ *the matrix* M_n^* *for* $n = q^r$*.*

▶ **Definition 20.** Let $\mathcal{K} \subseteq \mathcal{S}(\varnothing)$ be a class of sets. The *q-power* $\mathcal{K}^q \subseteq \mathcal{S}(\varnothing)$ *of* $\mathcal{K}$ consists of all sets $\mathfrak{A} = ([q^r])$ such that $\mathfrak{B} = ([r]) \in \mathcal{K}$.

▶ **Theorem 21.** *Let* $\mathcal{K} \subseteq \mathcal{S}(\varnothing)$*. If* $\mathcal{K}^q$ *is definable in* FOS_p*, then* $\mathcal{K}$ *is definable in* FOR_p*.*

Proof. If $\mathcal{K}^q$ is FOS_p-definable, then by Theorem 18 by a formula $\varphi = (\mathsf{slv}_p\, \bar{x}_1, \bar{x}_2)\alpha(\bar{x}_1, \bar{x}_2) \in \mathrm{FOS}_p$ where α is quantifier-free.

By using the above construction and Lemma 19, we conclude that the linear equation system $M_n \cdot \vec{x} = \mathbb{1}$ defined by α in an input structure $\mathfrak{A} = ([n])$ of size $n = q^r$ can be transformed into the equivalent system $M_n^* \cdot \vec{x} = \mathbb{1}$ which is FOC-definable in $\mathfrak{B} = ([r])$. Let $\varphi^* \in \mathrm{FOR}_p$ be a formula which expresses the solvability of the linear system $M_n^* \cdot \vec{x} = \mathbb{1}$ in a structure $\mathfrak{B} = ([r])$. Then $\mathfrak{B} \models \varphi^*$ if, and only if, $\mathfrak{A} \models \varphi$ since the linear equation systems $M_n \cdot \vec{x} = \mathbb{1}$ and $M_n^* \cdot \vec{x} = \mathbb{1}$ are equivalent. Hence φ^* defines $\mathcal{K}$. ◀

Proof of Theorem 17. Otherwise we would have $\mathrm{FOS}_p = \mathrm{FOR}_p$. Let $\mathcal{K} \subseteq \mathcal{S}(\varnothing)$ be a class of sets such that $\mathcal{K} \notin \mathrm{FOR}_p$, but such that $(\mathcal{K}^q)^q \in \mathrm{FOR}_p$. Such a class $\mathcal{K}$ is well-known to exist (just combine the fact that, over sets, we have $\textsc{Logspace} \leq \mathrm{FOR}_p \leq \textsc{Ptime}$ and the space-hierarchy theorem). Since $\mathrm{FOS}_p = \mathrm{FOR}_p$ we had $(\mathcal{K}^q)^q \in \mathrm{FOS}_p$ and by Theorem 21 this means that $\mathcal{K}^q \in \mathrm{FOR}_p$. Again, since $\mathrm{FOR}_p = \mathrm{FOS}_p$, we had $\mathcal{K}^q \in \mathrm{FOS}_p$. A second application of Theorem 21 yields $\mathcal{K} \in \mathrm{FOR}_p$ which contradicts our assumptions. ◀

Finally we remark that, in the absence of counting, the same proof works for the extension of fixed-point logic by solvability quantifiers. The simple reason is that fixed-point operators do not increase the expressive power of first-order logic over the empty signature since all definable relations are composed from a constant-sized set of basic building blocks.

5 Discussion

We showed that the expressive power of rank operators over different prime fields is incomparable and we inferred that the version of rank logic FPR with a distinct rank operator rk_p for every prime $p \in \mathbb{P}$ fails to capture polynomial time. In particular our proof shows that FPR cannot express the uniform version of the matrix rank problem where the prime p is part of the input. Moreover, we separated rank operators and solvability quantifiers in the absence of counting.

Of course, an immediate question is whether the extension FPR^* of FPC by the uniform rank operator rk^* suffices to capture polynomial time. We do not believe that this is the case. A natural candidate to separate FPR^* from $\textsc{Ptime}$ is the solvability problem for linear

equation systems over finite rings rather than fields [6]. While linear equations systems can be efficiently solved also over rings, there is no notion of matrix rank that seems to be helpful for this purpose. In particular, it is open whether FPR^* can define the isomorphism problem for CFI-structures generalised to $\mathbb{Z}_4$. A negative answer to this last question would provide a class of structures on which FPR^* is strictly weaker than Choiceless Polynomial Time (which captures PTIME on this class [1]).

Another question concerns the relationship between solvability logic FPS and rank logic FPR^*. Our proof of Lemma 7 shows that on every class of structures of bounded colour class size the two logics have the same expressive power. However, over general structures this reduction fails. We only know, by our results from Section 4, that a simulation of rank operators by solvability quantifiers would require counting.

Finally, we think it is worth to explore the connections between our approach and the game-theoretic approach proposed by Dawar and Holm in [8] to see to what extent our methods can be combined. For example, what kind of properties does a variant of their partition games have for infinitary logics with solvability quantifiers?

References

1 F. Abu Zaid, E. Grädel, M. Grohe, and W. Pakusa. Choiceless Polynomial Time on structures with small Abelian colour classes. In *MFCS 2014*, volume 8634 of *Lecture Notes in Computer Science*, pages 50–62. Springer, 2014.

2 A. Atserias, A. Bulatov, and A. Dawar. Affine systems of equations and counting infinitary logic. *Theoretical Computer Science*, 410:1666–1683, 2009.

3 G. Buntrock, U. Hertrampf, C. Damm, and C. Meinel. Structure and importance of logspace-mod-classes. *STACS'91*, pages 360–371, 1991.

4 J. Cai, M. Fürer, and N. Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, 1992.

5 A. Dawar. The nature and power of fixed-point logic with counting. *ACM SIGLOG News*, pages 8–21, 2015.

6 A. Dawar, E. Grädel, B. Holm, E. Kopczynski, and W. Pakusa. Definability of linear equation systems over groups and rings. *Logical Methods in Computer Science*, 9(4), 2013.

7 A. Dawar, M. Grohe, B. Holm, and B. Laubner. Logics with Rank Operators. In *LICS'09*, pages 113–122. IEEE Computer Society, 2009.

8 A. Dawar and B. Holm. Pebble games with algebraic rules. In *Automata, Languages, and Programming*, pages 251–262. Springer, 2012.

9 H.-D. Ebbinghaus and J. Flum. *Finite model theory*. Springer-Verlag, 2nd edition, 1999.

10 E. Grädel et al. *Finite Model Theory and Its Applications*. Springer, 2007.

11 E. Grädel and W. Pakusa. Rank logic is dead, long live rank logic! *CoRR*, abs/1503.05423, 2015.

12 M. Grohe. The quest for a logic capturing PTIME. In *LICS 2008*, pages 267–271, 2008.

13 Y. Gurevich and S. Shelah. On finite rigid structures. *The Journal of Symbolic Logic*, 61(02):549–562, 1996.

14 B. Holm. *Descriptive complexity of linear algebra*. PhD thesis, Univ. of Cambridge, 2010.

15 B. Laubner. *The structure of graphs and new logics for the characterization of Polynomial Time*. PhD thesis, Humboldt-Universität Berlin, 2011.

16 M. Otto. *Bounded Variable Logics and Counting*. Springer, 1997.

17 W. Pakusa. Finite model theory with operators from linear algebra. Staatsexamensarbeit, RWTH Aachen University, 2010.

18 J. Torán. On the hardness of graph isomorphism. *SIAM Journal on Computing*, 33(5):1093–1108, 2004.

Two-Restricted One Context Unification is in Polynomial Time*

Adrià Gascón[1], Manfred Schmidt-Schauß[2], and Ashish Tiwari[1]

1 **SRI International, Menlo Park, CA, USA**
adriagascon@gmail.com,tiwari@csl.sri.com
2 **Goethe-Universität, Frankfurt, Germany**
schauss@ki.informatik.uni-frankfurt.de

Abstract

One Context Unification (1CU) extends first-order unification by introducing a single context variable. This problem was recently shown to be in NP, but it is not known to be solvable in polynomial time. We show that the case of 1CU where the context variable occurs at most twice in the input (1CU2r) is solvable in polynomial time. Moreover, a polynomial representation of all solutions can be computed also in polynomial time. The 1CU2r problem is used as a subroutine in polynomial time algorithms for several more general classes of 1CU. Our algorithm can be seen as an extension of the usual rules of first-order unification and can be used to solve related problems in polynomial time, such as first-order unification of two terms that tolerates one clash, and several interesting classes of the general 1CU problem. All our results assume that the input terms are represented as Directed Acyclic Graphs.

1998 ACM Subject Classification F.3.2 Semantics of Programming Languages, F.4.1 Mathematical Logic, F.4.2 Grammars and Other Rewriting Systems

Keywords and phrases context unification, first-order unification, deduction, type checking

Digital Object Identifier 10.4230/LIPIcs.CSL.2015.405

1 Introduction

The well-known first-order unification problem consists of solving equations between terms with leaf variables ranging over terms. Context unification (CU) extends first-order unification by introducing context variables of arity one standing for contexts. Hence, the CU equations may contain subterms of the form $F(s)$, where F is a context variable that may be instantiated by a context. For example, the equation $F(a) \doteq f(a, a)$ has two solutions, since applying either substitution $\{F \to f(a, \bullet)\}$ or $\{F \to f(\bullet, a)\}$, gives the trivial equation $f(a, a) \doteq f(a, a)$.

Context unification falls in between first-order unification, which is solvable in linear time [19], and higher-order unification, which is undecidable [10]. The best known upper bound for the complexity of context unification is PSPACE [12]. Moreover, several variants and specializations of context unification have also been studied [16, 20, 15, 14, 6, 17, 7]. This paper is concerned with a particular case of CU called the *One Context Unification (1CU)* problem. In 1CU, only one context variable occurs in the input terms, possibly with

* This work was sponsored, in part, by National Science Foundation under grants CCF-1423296 and CNS-1423298, and ONR under subaward 60106452-107484-C under prime grant N00014-12-1-0914. The views, opinions, and/or findings contained in this report are those of the authors and should not be interpreted as representing the official views or policies, either expressed or implied, of the funding agencies.

many occurrences. This problem was recently proved to be in NP [8], but whether it is NP-hard or solvable in polynomial time is still open. That also holds for the case where the input terms are represented with Singleton Tree Grammars [4], a compression mechanism for terms that is more general than Directed Acyclic Graphs (DAGs). The initial interest in one context unification comes from interprocedural program analysis [11, 5], where context variables are used to represent (the yet unknown) summaries of procedures. In particular, one context unification problems (over uninterpreted terms) arise when analyzing programs using an abstract domain consisting of (uninterpreted) terms.

An interesting simple variant of the 1CU problem is term unification up to one clash, or fault tolerant unification. For example, while $f(x, f(a, y))$ unifies with $f(f(a, b), x)$, changing an a into a b in one of these two terms makes them non-unifiable. The terms $f(x, f(a, y))$ and $f(f(b, b), x)$ are, however, almost unifiable: if we are willing to accept disagreement at one position, namely position 2.1, the rest of the terms unify. The position 2.1 is a *distinguishing position* of these two terms. Note that two first-order terms s, t almost unify if the restricted 1CU instance $F(z_1) = s, F(z_2) = t$ has a solution, where z_1, z_2 do not occur in s, t. Intuitively, the position of the hole of $F\sigma$ in the solution indicates the distinguishing position in fault tolerant unification. A particular application of fault tolerant unification is Milner-polymorphic type checking, where in case the type check/computation fails, a most probable reason for the fail can be computed and presented to the programmer.

In this paper, we consider the special case of the one context unification problem consisting of only two equations $F(r_1) \doteq s_1, F(r_2) \doteq s_2$, where r_1, r_2, s_1, s_2 are first-order terms without occurrences of F, which we call 2-restricted 1CU (1CU2r). We present a polynomial time algorithm for deciding 1CU2r. We also show that a representation of all solutions can be constructed in polynomial time, which can be used to effectively enumerate all unifiers of the problem. All our results hold also when the input terms are represented as DAGs.

The restriction to two equations (1CU2r) is motivated by two facts. First, the solution to the 1CU2r problem also solves the problem of finding (all) distinguishing positions for two given first-order terms. Second, in a recent paper [9], we presented polynomial time algorithms for several classes of 1CU problem (that have more than 2 equations, but have other restrictions on the terms), but the algorithm in [9] *assumes* that 1CU2r can be efficiently solved. The result in [9] can be interpreted as showing that the 1CU2r problem is the "hard" part in solving the general 1CU problem, and it possibly exhibits several of the intricacies that make 1CU challenging. Our results, combined with [9], may open a way to construct a polynomial time algorithm for the unrestricted one context unification problem.

2 Preliminaries

We assume knowledge of first order terms, substitutions, and first-order unification (see [3, 1, 2]), and use the following notation: $\mathcal{F}$ denotes a fixed finite ranked alphabet, $\mathcal{X}$ is a set containing first-order variables and exactly one context variable F, f, g denote function symbols, a, b denote constant symbols, x, y, z denote (first-order) variables, and p, q denote positions (sequences of positive integers) in terms. Our algorithm introduces fresh first-order variables from a set $\mathcal{Y}$, which we denote by y with possible subindexes. We denote $\mathcal{X} \cup \mathcal{Y}$ as $\mathcal{V}$. Hence, we will argue about terms in $\mathcal{T}(\mathcal{F}, \mathcal{X})$, $\mathcal{T}(\mathcal{F}, \mathcal{V})$ and $\mathcal{T}(\mathcal{F}, \mathcal{X}) \cup \mathcal{Y}$.

The set of positions of a term t, denoted by $\mathtt{pos}(t)$, is defined recursively as $\mathtt{pos}(f(t_1, \ldots, t_m)) = \{\lambda\} \cup \{i.p \mid i \in \{1, \ldots, m\} \wedge p \in \mathtt{pos}(t_i)\}$. The *length* of a position is denoted by $|p|$. The *subterm* of a term t at a position $p \in \mathtt{pos}(t)$, denoted $t|_p$, is defined recursively as $t|_\lambda = t$ and $f(t_1, \ldots, t_m)|_{i.p} = t_i|_p$. With $<$ we denote the prefix relation among positions

and with $\prec$ the subterm relation among terms. We say that two positions are *parallel* if they are incomparable by the prefix relation. We also define $\texttt{topsymbol}(f(t_1,\ldots,t_n) = f$, and $\texttt{topsymbol}(x) = x$, and $\texttt{vars}(t) = \{x \in \mathcal{V} \mid t|_p = x\}$.

By $\texttt{maxarity}$, we denote the maximum arity of the symbols in $\mathcal{F}$, and by $\texttt{pos}(\mathcal{F})$, we denote the set of positions $\{\lambda\} \cup \{1,\ldots,\texttt{maxarity}\}^+$. We also use contexts C, D, where $\texttt{hp}(C)$ is the notation for the position of the hole, denoted $\bullet$. Analogously as for terms, we refer to contexts in $\mathcal{C}(\mathcal{F},\mathcal{X})$ and $\mathcal{C}(\mathcal{F},\mathcal{V})$. In the notation that mixes first-order terms, contexts and plugging terms into holes, we write $C[s]$ for the term where s is plugged into the hole of C, and CD for the context $C[D]$. Similarly, we write $F(s)$ for the application of the context variable F to the term s. Sometimes we will omit brackets, if this does not lead to confusion. Also, we denote by $t[s]_p$, the term obtained from t by replacing its subterm at position p by s. The *exponentiation* of a position p to a natural number n, denoted p^n, is the position recursively defined as $p^n = p.p^{n-|p|}$ if $n > |p| > 0$, and as $p^n = p_1$ if $n \leq |p|$, $p = p_1.p_2$, and $|p_1| = n$. Note that $p^0 = \lambda$. The *exponentiation* of a context C to a natural number n, denoted C^n, is defined analogously to p^n.

In this work, we deal with *equations on terms*, denoted by e with possible subindexes. Given an equation $e = (s \doteq t)$, we call the set $\{s,t\}$ the *topterms of* e, denoted $\texttt{topterms}(e)$. Similarly, for a set Δ of equations, $\texttt{topterms}(\Delta)$ denotes $\bigcup_{e\in\Delta}(\texttt{topterms}(e))$. Similarly, $\texttt{topvars}(e) = \texttt{topterms}(e) \cap \mathcal{V}$. By $|\Delta|$ we denote the number of equations in Δ.

A *substitution*, denoted by σ, θ, η, is a total function $\sigma : \mathcal{V} \to \mathcal{T}(\mathcal{F},\mathcal{V}) \cup \mathcal{C}(\mathcal{F},\mathcal{V})$ such that $\alpha\sigma \in \mathcal{T}(\mathcal{F},\mathcal{V})$ if α is a first-order variable and $\alpha\sigma \in \mathcal{C}(\mathcal{F},\mathcal{V})$ if α is a context variable. Substitutions are extended, in the usual way, to be mappings from terms to terms and contexts to contexts. We also extend the notion of application of a substitution to equations as $(s \doteq t)\sigma = (s\sigma \doteq t\sigma)$, to sets as $\Delta\sigma = \biguplus_{e\in\Delta}\{e\sigma\}$, and to pairs as $\langle r_1, r_2\rangle\sigma = \langle r_1\sigma, r_2\sigma\rangle$. The *domain of a substitution* σ, denoted $\texttt{dom}(\sigma)$, is defined as usual, i.e. $\texttt{dom}(\sigma) = \{z \in \mathcal{V} \mid z\sigma \neq z\}$. The *composition* of σ and θ, denoted $\theta \circ \sigma$, is defined as $\{\alpha \mapsto \alpha\sigma\theta \mid \alpha \in \texttt{dom}(\sigma) \cup \texttt{dom}(\theta)\}$. For substitutions σ, θ, $\sigma = \theta$ holds if $\forall z \in \mathcal{V} : z\sigma = z\theta$. Moreover, σ is *more general* than θ, denoted $\sigma \leq \theta$, if there exists η such that $\theta = \eta \circ \sigma$.

A *unifier* of two terms s, t is a substitution σ such that $s\sigma = t\sigma$. A unifier does not always exist. We capture that situation by simply saying that the unifier of s and t is $\bot$. We define the *most general unifier* of two *first-order* terms s and t, denoted $\texttt{mgu}(s = t)$, as *any* unifier σ of s and t such that, for every unifier θ of s and t, $\sigma \leq \theta$ holds. If such substitution does not exist we say that $\texttt{mgu}(s = t)$ is *not defined*, denoted $\texttt{mgu}(s = t) = \bot$. In an abuse of notation, we assume that $t\sigma = \bot$ for every term t if $\sigma = \bot$ and extend the definitions for the application of a substitution on a term, equation, set of equations, and list of terms accordingly.

First-order unification can be performed in polynomial time, if the algorithm works on a shared term representation. We assume that terms are represented as DAGs, which allows sharing of common subterms. When using DAGs, performing unification and applying substitutions takes polynomial time. However, for example visiting all positions in term may take worst-case exponential time.

3 One Context Unification

The One Context Unification Problem (1CU) consists on finding a unifier for a set of equations over first-order terms that are extended with a single context variable F. It is known that the problem is in NP. It is also known that, if the set of equations contains an equation of the form $F(r_1) \doteq CF(r_2)$ where C is not the empty context, then unification can be performed

in polynomial time. Thus, without loss of generality, we can focus on 1CU instances of the form

$$\{\ F(r_1) \doteq s_1,\ \ \ldots,\ \ F(r_n) \doteq s_n\ \}$$

where s_i and r_i do not contain occurrences of F, for all $i \in \{1, \ldots, n\}$.

▶ **Definition 3.1.** A 2-restricted 1CU instance $\mathcal{I}$ (referred to by 1CU2r) consists of two equations of the form

$$F(r_1) \doteq s,\ F(r_2) \doteq t$$

where F is a context variable and s, t, r_1, r_2 are terms that may contain first-order variables but not the context variable.

The size of $\mathcal{I}$, denoted $||\mathcal{I}||$, is the size of the DAG representing all the terms (including subterms) in $\mathcal{I}$. In other words, $||\mathcal{I}|| = |\mathtt{subterms}(\mathcal{I})|$. From now on, when we refer to a polynomial time algorithm for 1CU2r, we implicitly assume this measure.

A solution, or unifier, σ of a 1CU2r instance can be characterized by $\mathtt{hp}(F\sigma)$ – the position of the hole in $F\sigma$.

▶ **Example 3.2.** The instance $\mathcal{I} = \{F(a) \doteq f(x_0, x_0),\ F(b) \doteq f(f(x_1, x_1), f(a, b))\}$ has a solution $\sigma = \{x_0 \mapsto f(a, a),\ x_1 \mapsto a,\ F \mapsto f(f(a, a), f(a, \bullet))\}$. Here, $\mathtt{hp}(F\sigma) = 2.2$. There is another solution with hole position 1.1.

In the first part of this paper we solve the decision version of the 1CU2r problem and later we show how to compute a representation for all unifiers.

4 The Decision Version of 1CU2r

We present a polynomial time algorithm that decides the 1CU2r unification problem. We describe our algorithm using *inference rules* that operate on *states*. Starting from an *initial state* that describes the input 1CU2r instance, the algorithm works by repeatedly applying the inference rules until a *final* state (where no rule is applicable) is reached. The final state will be a special fail state if the input problem has no unifiers.

4.1 Defining the state

Our inference rules operate on states (configurations), which are tuples of the form (R, Δ), where R is a pair of terms and Δ is a set of equations $s_1 \doteq t_1, \ldots, s_n \doteq t_n$, where each equation can be asymmetric (unmarked) or symmetric (marked with a superscript S).

▶ **Definition 4.1** (Initial state)**.** For a given instance $\mathcal{I} = \{F(r_1) \doteq s, F(r_2) \doteq t\}$ of the 1CU2r problem, the initial state of our algorithm is $\mathcal{S}_0 = (\langle r_1, r_2\rangle, \{s \doteq t\})$.

All terms in a state are first-order terms. The terms in R are the left-hand side terms and the equations in Δ are (different possible) right-hand side terms. While $u \doteq v$ and $v \doteq u$ are different equations, we do not distinguish between $u \doteq^S v$ and $v \doteq^S u$. If we write $x = s$ without a dot, then this is only a notation for "either $x \doteq s$ or $x \doteq^S s$".

The reason for keeping the right-hand side terms s, t as an equation $s \doteq t$ is that our inference rules will be trying to (almost) first-order unify s and t, but for one position (which will be the hole position of F in the solution).

The mapping from a state to 1CU2r instances is defined as follows.

▶ **Definition 4.2.** Let $\mathcal{S} = (\langle r_1, r_2\rangle, \Delta)$ be a state, and let $e = (s =^S t) \in \Delta$ be a *symmetric* equation. Let $\theta = \mathtt{mgu}(\Delta \setminus \{e\})$. We define the two 1CU instances spanned by e in $\mathcal{S}$, denoted $P(e, \mathcal{S}, 1)$ and $P(e, \mathcal{S}, 2)$ (or simply $P(e, i)$ if $\mathcal{S}$ is clear from the context), as

$$\begin{array}{rcll} P(e,\mathcal{S},1) & = & \{F(r_1\theta) \doteq s\theta,\ F(r_2\theta) \doteq t\theta\} & \text{if } \theta \neq \bot \\ P(e,\mathcal{S},2) & = & \{F(r_2\theta) \doteq s\theta,\ F(r_1\theta) \doteq t\theta\} & \text{if } \theta \neq \bot \\ P(e,\mathcal{S},i) & = & \bot & \text{if } \theta = \bot, \text{ for } i = 1,2 \end{array}$$

For an asymmetric equation $e \in \Delta$, $P(e, \mathcal{S}, 1)$ is defined as it is defined for symmetric equations, but $P(e, \mathcal{S}, 2) = \bot$ always. Given a state $\mathcal{S} = (L, \Delta)$ and a subset $\Gamma \subseteq \Delta$, by $\mathtt{instances}(\Gamma)$ we denote the set $\bigcup_{e\in\Gamma}(\{P(e, \mathcal{S}, 1), P(e, \mathcal{S}, 2)\})$.

Note that every equation e in the set Δ of a state spans zero, one or two 1CU2r instances, depending on whether $\theta = \bot$ and whether e is symmetric. Note that the initial state for a 1CU2r instance spans that 1CU2r instance. We say that a state has a solution if one of the instances that it spans is unifiable.

4.2 An Illustrative Example

We illustrate our procedure on a simple example before presenting it formally. Consider the instance $\mathcal{I}$ from Example 3.2. The corresponding initial state is

$$\mathcal{S}_0 \quad = \quad (\langle a, b\rangle, \{f(x_0, x_0) \doteq f(f(x_1, x_1), f(a, b))\})$$

The idea behind our procedure is that it searches for a solution σ by searching for $\mathtt{hp}(F\sigma)$ – call it p. For this example, the value $p = \lambda$ (i.e., $F \mapsto \bullet$) does not work, and hence, we need to find if $p = 1.p'$ or $p = 2.p'$ for some p'. So, we "decompose" (as in first-order unification) the equation in Δ to get a new state

$$\mathcal{S}_1 \quad = \quad (\langle a, b\rangle, \{x_0 \doteq f(x_1, x_1),\ x_0 \doteq f(a, b)\})$$

Let us construct the two instances $\mathcal{I}_1, \mathcal{I}_2$ corresponding to the state $\mathcal{S}_1$.

$$\begin{array}{rcl} \mathcal{I}_1 & = & P(x_0 \doteq f(x_1, x_1), 1) \ = \ \{F(a) \doteq f(a, b),\ F(b) \doteq f(x_1, x_1)\} \\ \mathcal{I}_2 & = & P(x_0 \doteq f(a, b), 1) \ = \ \{F(a) \doteq f(x_1, x_1),\ F(b) \doteq f(a, b)\} \end{array}$$

Note that $\mathcal{I}$ has a solution iff $\mathcal{I}_1$ or $\mathcal{I}_2$ has a solution.

A natural way to proceed would be to solve $\mathcal{I}_1$ and $\mathcal{I}_2$ recursively. However, that approach may perform an exponential number of steps since it is visiting all positions in a term that is represented as a DAG. Instead, our algorithm, roughly speaking, solves $\mathcal{I}_1$ and $\mathcal{I}_2$ simultaneously using a "Merge rule", which generates state $\mathcal{S}_2$ from the state $\mathcal{S}_1$.

$$\mathcal{S}_2 \quad = \quad (\langle a, b\rangle, \{f(x_1, x_1) \doteq^S f(a, b)\}).$$

Note that, again by Definition 4.2, $\mathcal{S}_2$ spans $\mathcal{I}_1$ and $\mathcal{I}_2$, since $P(f(x_1, x_1) \doteq^S f(a, b), \mathcal{S}_2, 1) = \mathcal{I}_2$ and $P(f(x_1, x_1) \doteq^S f(a, b), \mathcal{S}_2, 2) = \mathcal{I}_1$, and hence we have not lost any solutions. Note that the symmetric mark indicates that we need to try both orientations of the equation.

We can continue by applying "decompose" to $\mathcal{S}_2$ to get $\mathcal{S}_3$:

$$\mathcal{S}_3 \quad = \quad (\langle a, b\rangle, \{x_1 \doteq^S a, x_1 \doteq^S b\}).$$

And we can again use the "Merge rule" to get $\mathcal{S}_4$:

$$\mathcal{S}_4 \quad = \quad (\langle a, b\rangle, \{a \doteq^S b\}).$$

$$\texttt{Decompose:}\quad \frac{\langle R, \Delta = \Gamma \cup \Delta' \rangle}{\texttt{SolveFO}(\mathcal{I}_1\sigma) \mid \ldots \mid \texttt{SolveFO}(\mathcal{I}_{2|\Gamma|}\sigma) \mid \texttt{Decompose}(\langle R, \Delta\rangle, \Gamma, \mathcal{Y})}$$
where $\bigcup \mathcal{I}_i = \texttt{instances}(\Gamma)$, $\sigma = \{F \to \bullet\}$, $|\texttt{topsymbols}(\Gamma)| = 1$, and $\Gamma \subset \Delta$ is a root class.

$$\texttt{Easy:}\quad \frac{\langle R, \Delta \cup \{e\}\rangle}{\texttt{SolvEz}(P(e,1)) \mid \texttt{SolvEz}(P(e,2)) \mid \langle R\,\texttt{mgu}(e), \Delta\,\texttt{mgu}(e)\rangle} \quad \text{if } P(e,1) \text{ is easy}$$

$$\texttt{InvEq:}\quad \frac{\langle R, \Delta\rangle}{\langle R\sigma, \Delta\sigma\rangle} \quad \text{if } \Delta \models (s = t) \text{ and } \sigma = \texttt{mgu}(s \doteq t)$$

$$\texttt{DiscardEq:}\quad \frac{\langle R, \Delta \cup \{e\}\rangle}{\langle R\,\texttt{mgu}(e), \Delta\,\texttt{mgu}(e)\rangle} \quad \text{if } \texttt{mgu}(\Delta) = \bot$$

$$\texttt{Merge:}\quad \frac{\langle R, \Delta \cup \{x = s, x = t\}\rangle}{\langle R, \Delta \cup \{s \doteq^S t\}\rangle} \quad \text{if } x \notin \texttt{vars}(\Delta) \text{ and } x \notin \texttt{vars}(R)$$

$$\texttt{Merge:}\quad \frac{\langle R, \Delta \cup \{x = s, t = x\}\rangle}{\langle R, \Delta \cup \{t \doteq^? s\}\rangle} \quad \text{if } x \notin \texttt{vars}(\Delta) \text{ and } x \notin \texttt{vars}(R)$$
where ? is S iff $x = s$ or $t = x$ is symmetric.

$$\texttt{Fail:}\quad \frac{\langle \bot, \bot\rangle}{fail}$$

Figure 1 Inference rules for deciding 2-restricted one context unification.

We observe that the instance $P(a \doteq^S b, \mathcal{S}_4, 1)$ has a unifier that maps the context variable to $\bullet$, and we can terminate declaring that the original problem $\mathcal{I}$ has a unifier.

Instead of $\mathcal{I}$, if we start with the instance $\mathcal{I}' = \{F(a) \doteq f(x_0, f(a,b)), F(b) \doteq f(f(x_1, x_1), x_0)\}$, then after decomposing the initial state S_0' we would get the state $\mathcal{S}_1'$ containing the equations $x_0 \doteq f(x_1, x_1),\ f(a,b) \doteq x_0$ in its Δ component. Now, the two instances $\mathcal{I}_1', \mathcal{I}_2'$ corresponding to this new state would be

$$\mathcal{I}_1' = P(x_0 \doteq f(x_1, x_1), \mathcal{S}_1', 1) = \{F(a) \doteq f(a,b),\ F(b) \doteq f(x_1, x_1)\}$$
$$\mathcal{I}_2' = P(f(a,b) \doteq x_0, \mathcal{S}_1', 1) = \{F(a) \doteq f(a,b),\ F(b) \doteq f(x_1, x_1)\}$$

These two instances are identical! In this case, we again apply "merge" on $\mathcal{S}_1'$, but we now get a new state $\mathcal{S}_2'$ with an *asymmetric* equation:

$$\mathcal{S}_2' = (\langle a, b\rangle, \{f(a,b) \doteq f(x_1, x_1)\}).$$

We "decompose" $\mathcal{S}_2'$ to get $\mathcal{S}_3'$ with equations $a \doteq x_1, b \doteq x_1$ (both asymmetric this time), but when we then apply "merge" on $\mathcal{S}_3'$, since x_1 is on the same side of the two equations, we generate a *symmetric* equation in $\mathcal{S}_4'$:

$$\mathcal{S}_4' = (\langle a, b\rangle, \{a \doteq^S b\})$$

Since one of the instances spanned by $\mathcal{S}_4'$ has a unifier, we know $\mathcal{I}'$ too has a unifier.

4.3 Inference Rules for Deciding 1CU2r

The inference rules for solving the decision version of 1CU2r are presented in Figure 1. The rule `Decompose` and the rule `Easy` generate more than one state (separated by |). However, all but one of the child states is immediately evaluated to $\top$ or $\bot$, and there is only one main child (shown last) along which the search proceeds (if all others evaluate to $\bot$).

Since we are working on a DAG representation for Δ, it is useful to keep in mind the graph representing the set Δ.

▶ **Definition 4.3.** Let $\mathcal{S} = (R, \Delta)$ be a state. Let V be the nodes of the DAG representing the terms (and subterms) in Δ. By $\equiv$ we denote the set of undirected edges $\{(u, v) \mid \exists u = v \in \Delta\}$. By $\rightarrow$ we denote the set of subterm edges $\{(u, v) \mid u = f(\ldots, v, \ldots) \text{ for some } f\}$. By $G(\Delta)$ we denote the graph with nodes V and the two kinds of edges $\equiv$ and $\rightarrow$.

We describe the individual inference rules below.

4.3.1 The `Decompose` rule

As mentioned above, the `Decompose` rule generates several states. However, all of them but one correspond to first-order unification instances that are solved immediately in polynomial time using a procedure `SolveFO`.

As in the Paterson and Wegman linear unification [19] algorithm, the `Decompose` rule first finds a *root class* $\Gamma \subseteq \Delta$ in the graph $G(\Delta)$ (Definition 4.3). A root class is a maximum cardinality subset Γ of Δ such that if t is a topterm in Γ, then (a) t is not a proper subterm of any term in Δ, and (b) t is not a topterm in $\Delta - \Gamma$. If such a nonempty $\Gamma = \{e_1, \ldots, e_{|\Gamma|}\}$ exists, and if all non-variable terms in Γ have top symbol f, then the `Decompose` rule can be applied. First, we instantiate the variables in Γ. Let $\theta = \{x \rightarrow f(y^x_1, \ldots, y^x_{ar(f)}) \mid x \in \texttt{topvars}(\Gamma), y^x_i \text{ are fresh variables from } \mathcal{Y}\}$. Note that all terms in $\Gamma\theta$ will have f as their top symbol.

▶ **Definition 4.4** (Decompose). Given a state $\mathcal{S} = (R, \Delta)$, we define $\texttt{Decompose}(\mathcal{S}, \Gamma, \mathcal{Y})$ as the state $\mathcal{S}' = (R\theta, \Delta')$ where $\Delta' = (\Delta - \Gamma) \cup \Gamma'$ and Γ' is obtained from $\Gamma\theta$ as follows:

1. If there is an asymmetric equation $e = (f(u_1, \ldots, u_k) \doteq f(v_1, \ldots, v_k))$ in $\Gamma\theta$, then replace e by several equations $u_1 \doteq v_1, \ldots, u_k \doteq v_k$.
2. If there is a symmetric equation $e = (f(u_1, \ldots, u_k) \stackrel{S}{\doteq} f(v_1, \ldots, v_k))$ in $\Gamma\theta$, then replace e by several equations $u_1 \stackrel{S}{\doteq} v_1, \ldots, u_k \stackrel{S}{\doteq} v_k$.

Note the differences with the decompose rule in classical first-order unification: in first-order unification, decomposition is not performed when there are variables in an equivalence class. Moreover, we restrict our application of the rule to a root class as in Paterson and Wegman [19]. Conditions (a) and (b) above ensure the invariant that variables from $\mathcal{Y}$ occur only as topterms in the equations Δ. This fact will be crucial to prove termination in polynomial time.

4.3.2 The Shrink rules

The rules `Easy`, `InvEq`, and `DiscardEq` remove at least one equation from the set Δ and help in guaranteeing that the size of Δ stays polynomial w.r.t. the input size.

4.3.2.1 The `Easy` rule

We first enumerate some easily solvable instances of 1CU2r.

▶ **Definition 4.5** (Easy). A 1CU2r instance $\{F(r_1) \doteq s, F(r_2) \doteq t\}$ is *easy* if either one of the following condition holds: (a) $t = C[s]$ (or $s = C[t]$) for some non-empty context C, or (b) $\texttt{topsymbol}(s) \neq \texttt{topsymbol}(t)$, or (c) $s = t$, or (d) $s \in \mathcal{X}$ or $t \in \mathcal{X}$, or (e) s (or t) occurs (as a subterm) in r_1 or r_2.

▶ **Lemma 4.6** (Easy). *There is a polynomial time procedure* `SolvEz` *that returns* $\top$ *(denoting* `True`*) iff a given easy instance* $\mathcal{I}$ *is unifiable.*

We will discuss easy instances in more detail in Section 6. For a symmetric equation e, note that $P(e,1)$ is easy iff $P(e,2)$ is easy. The rule `Easy` checks if $P(e,1)$ is easy for some $e \in \Delta$ and solves it immediately. If both $P(e,i)$ are found to be non-unifiable, then the search proceeds with the new state obtained by removing e from Δ. Note that e is removed by applying $\texttt{mgu}(e)$ to the rest of Δ and R, and recall that if $\texttt{mgu}(e) = \bot$ then $\Delta\,\texttt{mgu}(e) = \bot$.

4.3.2.2 The `InvEq` rule

If there are terms s and t that are made equal by every unifier of a 1CU2r instance, then we can as well unify s and t and apply the unifier to the 1CU2r instance. The rule `InvEq` performs this step. Let $\mathcal{S} = (R, \Delta)$ be a state. We write $\Delta \models (s = t)$ if s and t are distinct terms that lie in an $\equiv$-cycle of $G(\Delta)$ (cycle consisting solely of $\equiv$ edges). It should be evident that if $\Delta \models (s = t)$, then for all $e \in \Delta$, $s\sigma = t\sigma$ holds for mgu σ of $\Delta \setminus \{e\}$; that is, s and t are made equal by every unifier of any instance spanned by the state $\mathcal{S}$.

4.3.2.3 The `DiscardEq` rule

The rule `DiscardEq` removes an equation e from Δ if the problems $P(e, \mathcal{S}, i)$ spanned by e have no solutions. This happens, for example, when $\texttt{mgu}(\Delta \setminus \{e\}) = \bot$, by Definition 4.2.

4.4 The `Merge` rule

Recall that the `Decompose` rule instantiates variables in the root class to enable the decomposition. In one special case, we prefer to apply the `Merge` rule rather than the `Decompose` rule. Specifically, the `Merge` rule is applicable to state (R, Δ) if the root class $\Gamma \subset \Delta$ contains a variable x such that (a) x occurs exactly twice in $\texttt{topterms}(\Gamma)$ and (b) x does not occur in the terms of R.

In such a case, $\Delta = \Delta' \cup \{e_1, e_2\}$, where e_1, e_2 might be of the form:

1. $e_1 = (x = s_1)$ and $e_2 = (x = s_2)$, or $e_1 = (s_1 = x)$ and $e_2 = (s_2 = x)$: In these cases, the two equations e_1, e_2 are replaced by a single symmetric equation $s_1 \doteq^S s_2$.
2. $e_1 = (s_1 = x)$ and $e_2 = (x = s_2)$: In this case, e_1, e_2 are replaced by $s_1 \doteq s_2$ if both e_1 and e_2 are asymmetric, and by $s_1 \doteq^S s_2$ otherwise.

The `Merge` rule captures one of the key insights in our procedure: we can simultaneously search for solutions for $\{F(r_1) \doteq s, F(r_2) \doteq t\}$ and $\{F(r_1) \doteq t, F(r_2) \doteq s\}$, and this commutativity is the only source of blowup for 1CU2r (see also the example in 4.2). While most of the rules presented so far have some similarity to the rules for more general 1CU problems that we presented in [9], there is no analogue of the merge rule in [9].

4.5 Correctness

We fix a strategy for applying the rules. Specifically, we apply the shrink rules and the `Merge` rules exhaustively (denoted by "!"), and apply `Decompose` only when none of the other rules are applicable.

$$((\text{Shrink} \mid \text{Merge})^!(\text{Decompose}))^!$$

To provide intuition for the working of our procedure, consider the graph $G(\Delta)$ with $\equiv$ and $\rightarrow$ edges (Definition 4.3). Either there is a cycle in this graph or there is none. If there is an $\equiv$-cycle, then we can apply `InvEq` rule to eliminate it. If there is a cycle that goes through an $\rightarrow$ (subterm) edge, (which corresponds to an occurs-check violation in first-order unification), then the inference rule `Easy` will be applicable (since for some edge

$s \doteq t$ in the cycle, we will have $s = C[t]$ or $t = C[s]$ for some nonempty context C, which is Case (a) in Definition 4.5). We exhaustively apply shrink rules, and since they reduce $|\Delta|$ or $|\texttt{topterms}(\Delta)|$, we will eventually stop. When none of the shrink rules are applicable, then it means there are no cycles in $G(\Delta)$. This implies that there will be a root class Γ. If there are two different top symbols, say f and g, among the terms in Γ, then we would apply the `Easy` rule first (some instance spanned by the state will satisfy Case (b) of Definition 4.5). If Γ contains only equations between variables, then these variables can not occur anywhere else in Δ (because Γ is the root class), and hence, again `Easy` would be applicable (some instance spanned by the state will satisfy Case (d) of Definition 4.5). Hence, we conclude that Γ contains at least one non-variable term and all non-variable terms will have the same function symbol at the top. As a result, the `Decompose` rule will be applicable. If `Merge` is applicable, we first apply that rule and finally, when no more `Merge` applications are possible, we apply `Decompose`, and repeat.

We say that a state has a solution if one of its spanned instances (Definition 4.2) has a solution. It is easily verified that each inference rule preserves the existence of a solution, and it does not introduce any new solutions. Furthermore, since the above argument guarantees progress (stated in Lemma 4.7), correctness follows.

▶ **Lemma 4.7.** *Let $\mathcal{S} = (R, \Delta)$ be a valid state of our algorithm. If $\Delta \neq \emptyset$, then a rule can be applied to $\mathcal{S}$.*

4.6 Termination

Without loss of generality, we assume that the DAG D representing all the terms (and subterms) in any state are minimal in size, and hence the correspondence between terms and nodes is bijective. Hence, we can then refer to nodes of D and subterms of the problem as if they were the same thing. In the rest of this section, when we obtain a bound w.r.t. $\texttt{subterms}(\texttt{topterms}(\Delta))$, for some set of equations Δ, the bound directly translates to the size of the DAG D representing the terms in Δ. A crucial observation is that the *number* of terms represented in a DAG is preserved by the application of substitutions resulting from the unification of terms of the DAG. This is because application of such substitutions can be achieved by manipulating only the *edges* of the DAG, leaving its nodes untouched.

Our algorithm reduces deciding solvability of a 1CU2r instance $\mathcal{I}$ to solving instances $\mathcal{I}_1, \ldots, \mathcal{I}_n$ where each $\mathcal{I}_i$ is either an *easy* instance (generated by rule `Easy`) or a first-order unification instance (generated by rule `Decompose`). Hence, to prove that our algorithm terminates in polynomial time we have to argue that (A) $\mathcal{I}_i$ has polynomial size, for all i, and that (B) n is polynomial.

One of the main difficulties in the proof is due to the fact our inference system introduces fresh variables from $\mathcal{Y}$. We first show that the rule applications do not increase the total number of subterms in the equations of Δ that are not variables from $\mathcal{Y}$. Moreover, such a measure decreases with every application of `Decompose`.

▶ **Lemma 4.8.** *Let $(R, \Delta_0) \to^* (R_k, \Delta_k)$ be a derivation starting from a valid initial state. Then, $|\texttt{subterms}(\texttt{topterms}(\Delta_k) \setminus \mathcal{Y})| \leq |\texttt{subterms}(\texttt{topterms}(\Delta_0)|$. Moreover, if $(R, \Delta_0) \to^* (R_k, \Delta_k) \to_{\texttt{Decompose}} (R_{k+1}, \Delta_{k+1})$ then $|\texttt{subterms}(\texttt{topterms}(\Delta_{k+1})) \setminus \mathcal{V}| < |\texttt{subterms}(\texttt{topterms}(\Delta_k)) \setminus \mathcal{V}|$.*

Proof. The lemma follows by inspecting the rules, the observation above that the total number of subterms from $\mathcal{T}(\mathcal{F}, \mathcal{X})$ in every Δ_i is preserved by the application of substitutions resulting from the unification of terms from $\mathcal{T}(\mathcal{F}, \mathcal{X})$, and the assumption, w.l.o.g., that

variables from $\mathcal{Y}$ are always instantiated in terms of variables from $\mathcal{X}$. The second part of the lemma follows from the maximality of Γ in the `Decompose` rule. ◀

Let us call a state minimized if the shrink and merge rules are not applicable. We now prove a bound on the cardinality of Δ in minimized states that is independent of the number of (old and new) variables in the equations.

▶ **Lemma 4.9.** *Let (R, Δ) be a minimized state. Then, $|\Delta| \leq 2|\mathit{topterms}(\Delta) \setminus \mathcal{V}|$.*

Proof. Consider the subgraph G' of $G(\Delta)$ (Definition 4.3) defined by only the $\equiv$ edges. Nodes of G' in $\mathcal{V}$ are called v-nodes, while the rest are called f-nodes. Note that, since Δ is minimized, the following holds: (i) G' is acyclic (by non-applicability of `Easy`), (ii) Every v-node in G' has degree at least 3 and each subtree hanging off v has at least one f-node (by non-applicability of `Easy` and `Merge`). We prove that for any forest that satisfies (i) and (ii), $n \leq 2m - 2$, where n is the number of nodes the forest and m is the number of f-nodes in it. For simplicity, and w.l.o.g., assume that G' only has one connected component and hence it is a tree. We use induction on m. (Base case) If $m = 2$, then $n = 2$ and the claim holds. (Induction step) Let $m > 2$. If there are no v-nodes in G', then $n = m > 2$ and the claim holds. So, let v be a v-node in G' connected to the disjoint subtrees $\{t_1, \ldots, t_l\}$. Note that $l \geq 3$. Let n_i and m_i be the number of nodes and f-nodes in tree t_i, respectively, for all i. Note that $m_i > 0$. Consider tree t_i with node v attached to it. Call it t'_i. If we treat v as an f-node in t'_i, then it satisfies (i) and (ii) and we can apply induction hypothesis to get $n_i + 1 \leq 2(m_i + 1) - 2$; that is, $n_i \leq 2m_i - 1$. Hence we have $n = \sum_{i=1}^{l}(n_i) + 1 \leq \sum_{i=1}^{l}(2m_i - 1) + 1 = 2m - l + 1 \leq 2m - 2$, where the last step holds because $l \geq 3$. Hence, since the total number of nodes in G' is less than $2|\mathtt{topterms}(\Delta) \setminus \mathcal{V}|$, so is its number of edges and thus $|\Delta|$. ◀

We can now extend the previous claim to any state. The following lemma relies on the rule application strategy.

▶ **Lemma 4.10.** *Let $(R_0, \Delta_0) \to^* (R_k, \Delta_k)$ be a derivation starting from a valid initial state. Then, $|\Delta_k| \leq 2|\mathit{topterms}(\Delta_k) \setminus \mathcal{V}|\mathtt{maxarity}$.*

Proof. We use induction on k. The lemma trivially holds for Δ_0, since it only contains one equation and we can assume w.l.o.g. that such equation contains a non-variable term. For the inductive step, note that the property of the lemma is trivially preserved by all the rules but `Decompose`, since such rules do not increase the size of the Δ component of the state. For the `Decompose` rule, recall that this rule is only applicable to a minimized state Δ. By Lemma 4.9, Δ satisfies $|\Delta| \leq 2|\mathtt{topterms}(\Delta) \setminus \mathcal{V}|$. Hence, the lemma follows from the fact that `Decompose` increases the number of equations in Δ by a factor of, at most, `maxarity`. ◀

We can finally prove claims (A) and (B) required for polynomial time termination.

▶ **Lemma 4.11.** *Let $\mathcal{I}$ be a 1CU2r instance and let $(R_0, \Delta_0) \to^* \mathcal{S}_k = (R_k, \Delta_k)$ be its corresponding derivation. Then, $k \leq 4||\mathcal{I}||^2\mathtt{maxarity}$ and, for every equation $e \in \Delta_k$, $P(e, \mathcal{S}_k, 1)$ and $P(e, \mathcal{S}_k, 2)$ have polynomial size with respect to $||\mathcal{I}||$.*

Proof. By Lemma 4.8, $\forall i \in \{1, \ldots, k\} : |\mathtt{subterms}(\mathtt{topterms}(\Delta_i) \setminus \mathcal{Y})| \leq |\mathtt{subterms}(\mathtt{topterms}(\Delta_0))|$, i.e., the number of non-variable subterms in the Δ_is in the derivation does not increase. By Lemma 4.10, we have that every Δ_i in the derivation satisfies $|\Delta_i| \leq 2|\mathtt{topterms}(\Delta_i) \setminus \mathcal{V}|\mathtt{maxarity}$. Since $|\mathtt{topterms}(\Delta_i) \setminus \mathcal{V}| \leq |\mathtt{subterms}(\mathtt{topterms}(\Delta_i) \setminus \mathcal{Y})|$, we

have the bound $|\Delta_i| \leq 2|\mathtt{subterms}(\mathtt{topterms}(\Delta_0))|\mathtt{maxarity} \leq 2||\mathcal{I}||\mathtt{maxarity}$. For the first statement of the lemma, note that subsequences of rule applications without the `Decompose` rule have length at most $2|\Delta_i| \leq 4||\mathcal{I}||\mathtt{maxarity}$, since the rules in such subsequences either reduce the cardinality of Δ or $\mathtt{topterms}(\Delta)$. On the other hand, by Lemma 4.8, each application of `Decompose` decreases the measure $|\mathtt{subterms}(\mathtt{topterms}(\Delta_i) \setminus \mathcal{Y})| \leq |\mathtt{subterms}(\mathtt{topterms}(\Delta_0))| \leq ||\mathcal{I}||$. Hence, we have $k \leq 4||\mathcal{I}||^2\mathtt{maxarity}$. The second statement of the lemma follows from the fact that k is polynomial and that DAGs are used for term representation. ◀

5 Representing All Unifiers

We now extend the procedure for deciding 1CU2r to also output a representation for all unifiers. The procedure works in the same way, except that now we do not terminate when we find an instance that is unifiable (in Rules `Decompose` and `Easy`), but rather we store the representation for all solutions returned by the subroutines and continue.

Given a 1CU2r instance $\mathcal{I}$, rather than returning all possible unifiers σ of $\mathcal{I}$, our algorithm will only return the positions $\mathtt{hp}(F\sigma)$. This is not a loss of generality, since σ can be recovered from $\mathtt{hp}(F\sigma)$ and $\mathcal{I}$ in polynomial time.

▶ **Definition 5.1** (Sets of solutions)**.** Let $\mathcal{I}$ be a 1CU2r instance. The set $\mathtt{HP}(\mathcal{I})$ of solutions of $\mathcal{I}$ is the set of positions $\{\mathtt{hp}(F\sigma) \mid \sigma \text{ is a solution of } \mathcal{I}\}$.

Using $\mathtt{HP}(\mathcal{I})$ as a substitute for all unifiers simplifies presentation and notation considerably. However, the cardinality of $\mathtt{HP}(\mathcal{I})$ may grow exponentially, or even be unbounded.

▶ **Example 5.2.** Let s be $f(x_0, x_0)$ and let $t_0 = f(a, b)$, $t_1 = f(f(x_1, x_1), f(a, b))$, and in general, let $t_n = f(f(x_n, x_n), t_{n-1})$ for $n \geq 1$. If $\mathcal{I} = \{F(z_1) \doteq s, F(z_2) \doteq t_n\}$ where z_1, z_2 are fresh, then the set $\mathtt{HP}(\mathcal{I})$ contains all positions with length at most $n + 1$, and hence, its cardinality is exponential in n. If $\mathcal{I} = \{F(a) \doteq s, F(b) \doteq t_n\}$, then $\mathtt{HP}(\mathcal{I})$ contains all positions of length n that contain an even number of 1s. These instances have an exponential worst-case running time for the algorithms in [8]. Our algorithm will represent the sets $\mathtt{HP}(\mathcal{I})$ in polynomial space by means of *abstract positions*.

Our algorithm will construct a succinct representation, using a grammar-formalism and also exponentiation, for $\mathtt{HP}(\mathcal{I})$, satisfying that: (1) a solution of $\mathcal{I}$ can be obtained in polynomial time, (2) the number of solutions m (or whether it is infinite) of $\mathcal{I}$ can be found (or decided) in polynomial time, and (3) all solutions of $\mathcal{I}$ can be obtained in polynomial time w.r.t. m.

To keep track of all solutions, the states over which our inference system works will be tuples of the form (R, Δ, S), where R is a pair of terms, Δ is a set of *labeled*, possibly marked (with S), equations $s_1 \doteq_{l_1} t_1, \ldots, s_n \doteq_{l_n} t_n$, where no label l_i occurs more than once, and S is a set of (*representations* for) set of positions. For an instance $\mathcal{I} = \{F(r_1) \doteq s, F(r_2) \doteq t\}$, the initial state will now be $(\langle r_1, r_2\rangle, \{s \doteq_\lambda t\}, \emptyset)$.

5.1 The Labels and the Representation of Solutions

The set L of labels in the equations Δ are terms generated using the following BNF grammar:

$$L \quad ::== \quad i \mid L.L \mid L + L \mid L \oplus L$$

where i denotes a number in $\{1, \ldots, \mathtt{maxarity}\}$. We call elements of L *abstract positions*. Note that L contains all concrete positions (that is, $\mathtt{pos}(\mathcal{F}) \subset L$). Each abstract position

denotes a set of concrete positions, but we delay the formal definition of the mapping until later. Intuitively, $\oplus$ and $+$ are used to distinguish applications of the `Merge` rule that produce a symmetric equation from those that do not. This information is necessary to correctly recover all solutions in our representation.

The elements of the third component of the state, which represent a set of solutions, are strings in the set

$$L_S = \{\langle l, i\rangle \mid l \in L, i \in \{1,2\}\}^*.\texttt{BaseCases} \cup \{\lambda\}$$

where the set `BaseCases` will be defined later (in Section 5.3 and Section 6). Since expanding the labels may lead to exponentially large labels, the algorithm in fact uses only the nonterminals of a dynamically extended Straight-Line Program (SLP), similarly as done in [13] (see [18] for a survey on algorithms on SLP-compressed words), which keeps the size polynomial by sharing common prefixes. For simplicity, we obviate this representation in the definitions of the rules.

We assume that, for the easy instances (Definition 4.5), we have a procedure that returns some representation of the set of all solutions, which we have denoted by `BaseCases` above.

5.2 Modified Inference Rules

For each inference rule in Figure 1, we have to show how we update the labels in Δ and the third component of the state.

5.2.1 Modification for the Third Component

The only rules that change the third component are `Decompose` and `Easy`. We just need to compute the solutions for the easily solvable instances generated by these two rules, and add these solutions to the third component of the main branch.

▶ **Rule 5.3** (`Decompose`). *When applying* `Decompose` *to a state* $\mathcal{S} = (R, \Delta = \Gamma \cup \Delta', S)$*, the function* $\texttt{Decompose}(\langle R, \Delta\rangle, \Gamma, \mathcal{Y})$ *returns the state whose third component* S' *is obtained as follows:*

$S' = S$;
for $(s =_l t) \in \Gamma$ ***do***
  Compute $\sigma_i = \texttt{mgu}(P(s =_l t, \mathcal{S}, i)\{F \to \bullet\})$*, for* $i \in \{1,2\}$;
  $S' = S' \cup \{\langle l, i\rangle \mid i \in \{1,2\}, \sigma_i \neq \bot\})$;
end for

Note that σ_i *corresponds to the* `mgu` *of the* i*th first-order unification problem resulting from the application of the* `Decompose` *rule in Figure 1.*

▶ **Rule 5.4** (`Easy`). *When applying* `Easy` *to a state* $\mathcal{S} = (R, \Delta \cup \{s \doteq_l t\}, S)$*, the third component* S' *of the last state generated by* `Easy` *is computed as follows:*

$$S' = S \cup \{\langle l, i\rangle.\texttt{SolvEz}(P(s \doteq_l t, \mathcal{S}, i)) \mid i \in \{1,2\}\}$$

where `SolvEz` *is now assumed to return a representation in* `BaseCases`*.*

5.2.2 Modifications for the Second Component

The only inference rules that add new equations in Δ are `Decompose` and `Merge`, and hence we need to specify how labels are assigned to these newly added equations. Labels on existing equations in Δ do not change.

The Decompose rule uses Definition 4.4 to generate the new state. In Definition 4.4, if the equation $e = (f(u_1, \ldots, u_k) \doteq_l f(v_1, \ldots, v_k))$ has label l, then the i-th generated equation, namely $u_i \doteq_{l.i} v_i$ is assigned label $l.i$. Labels on the symmetric variant in Definition 4.4 are assigned in the same way. The Merge rule introduces $+$ and $\oplus$ operators in the labels.

▶ **Rule 5.5** (Merge). *When applying* Merge *to a state* $\mathcal{S} = (R, \Delta \cup \{e_1, e_2\}, S)$*, if* e_1 *has label* l_1 *and* e_2 *has label* l_2*, then the generated equation has label* $l_1 \oplus l_2$ *if it is symmetric, and it has label* $l_1 + l_2$ *if it is asymmetric.*

The following example, derived from the family in Example 5.2 illustrates the goal of the Merge rule. Without it the algorithm would still be sound, but it would have worst-case exponential running time.

▶ **Example 5.6.** Consider the 1CU2r instance $\mathcal{I} = \{F(z_1) \doteq f(x_0, x_0), F(z_2) \doteq f(f(x_1, x_1), f(f(x_2, x_2), f(a, b)))\}$. The corresponding initial state of our algorithm is $\mathcal{S}_0 = (R = \langle z_1, z_2\rangle, \Delta = \{f(x_0, x_0) \doteq_\lambda f(f(x_1, x_1), f(f(x_2, x_2), f(a, b)))\}, S_0 = \emptyset)$, and we have the following derivation:
$\mathcal{S}_0 \to_{\texttt{Decompose}}$
$\mathcal{S}_1 = (R, \{x_0 \doteq_1 f(x_1, x_1), x_0 \doteq_2 f(f(x_2, x_2), f(a, b))\}, S_1 = \{\langle \lambda, 1\rangle\}) \to_{\texttt{Merge}}$
$\mathcal{S}_2 = (R, \{f(x_1, x_1) \doteq^S_{1\oplus 2} f(f(x_2, x_2), f(a, b))\}, S_1) \to_{\texttt{Decompose}}$
$\mathcal{S}_3 = (R, \{x_1 \doteq^S_{(1\oplus 2).1} f(x_2, x_2), x_1 \doteq^S_{(1\oplus 2).2} f(a, b)\}, S_2 = S_1 \cup \{\langle 1 \oplus 2, 1\rangle, \langle 1 \oplus 2, 2\rangle\}) \to_{\texttt{Merge}}$
$\mathcal{S}_4 = (R, \{f(x_2, x_2) \doteq^S_{((1\oplus 2).1)\oplus((1\oplus 2).2)} f(a, b)\}, S_2) \to_{\texttt{Decompose}}$
$\mathcal{S}_5 = (R, \{x_2 \doteq^S_{((1\oplus 2).1)\oplus((1\oplus 2).2).1} a, x_2 \doteq^S_{((1\oplus 2).1)\oplus((1\oplus 2).2).2} b\}, S_3 = S_2 \cup \{\langle (((1 \oplus 2).1) \oplus ((1 \oplus 2).2), 1\rangle, \langle (((1 \oplus 2).1) \oplus ((1 \oplus 2).2), 2\rangle\}) \to_{\texttt{Merge}}$
$\mathcal{S}_6 = (R, \{a \doteq^S_{(((1\oplus 2).1)\oplus((1\oplus 2).2).1)\oplus(((1\oplus 2).1)\oplus((1\oplus 2).2).1)} b\}, S_3) \to_{\texttt{Decompose}}$
$\mathcal{S}_7 = (R, \emptyset, S_4 = S_3 \cup \{\langle (((1 \oplus 2).1) \oplus ((1 \oplus 2).2).1 \oplus ((1 \oplus 2).1) \oplus ((1 \oplus 2).2).1, 1\rangle, \langle ((1 \oplus 2).1) \oplus ((1 \oplus 2).2).1 \oplus ((1 \oplus 2).1) \oplus ((1 \oplus 2).2).1, 2\rangle\})$

This example also illustrates the need for a succinct representation for the elements in the set of solutions of the state. As mentioned in Section 5.1, we use a common grammar-like representation G that shares common prefixes for that purpose. Instead of defining such representation formally, we just extend the previous example by showing how S_4 looks like with the suitable representation: $S_4 = \{\langle N_0, 1\rangle, \langle N_1, 1\rangle, \langle N_1, 2\rangle, \langle N_4, 1\rangle, \langle N_4, 2\rangle, \langle N_7, 1\rangle, \langle N_7, 2\rangle\}$, where G is the set of rules $\{N_0 \to \lambda, N_1 \to (1 \oplus 2), N_2 \to N_1.1, N_3 \to N_1.2, N_4 \to (N_2 \oplus N_3), N_5 \to N_4.1, N_6 \to N_4.2, N_7 \to (N_5 \oplus N_6)\}$. This representation also allows us to update the set of solutions after a rule application in polynomial time.

5.3 Correctness

The strategy for applying the inference rules is unchanged. Note that the elements of the third component of the state are strings in the set L_S (defined in Section 5.1). We define $\texttt{BaseCases} = \{\texttt{exp}(p, e), \texttt{exp}(p, k_1 N + k_2), \texttt{hps}(u, v), p.free\}$, for a concrete position $p \in \texttt{pos}(\mathcal{F})$, natural numbers $e, k_1, k_2 \leq ||\mathcal{I}||$, and an integer variable N. Note that the elements of BaseCases are simply symbolic expressions that are succinct representations for sets of positions (as defined in Figure 2). The *concretization function* $\alpha : L_s \to \mathcal{P}(\texttt{pos}(\mathcal{F}))$, presented in Figure 2, maps elements in the third component of the state to a set of positions.

Before we can state the correctness claim that solutions are preserved by inference rule applications, we need to define the set of solutions of a state.

▶ **Definition 5.7** (Solution of a state). Let $\mathcal{S} = (R, \Delta, S)$ be a state. The set of solutions of $\mathcal{S}$, denoted $\texttt{HP}(\mathcal{S})$, is defined as $\texttt{HP}(\mathcal{S}) = \bigcup_{(s=_l t)\in\Delta} \bigcup_{i\in\{1,2\}} (\alpha(\langle l, i\rangle).\texttt{HP}(P(s =_l t, \mathcal{S}, i)))$

$$
\begin{aligned}
&\alpha(\langle i,1\rangle) = \{i\}\\
&\alpha(\langle l_1.l_2,1\rangle) = \big(\alpha(\langle l_1,1\rangle).\alpha(\langle l_2,1\rangle)\big)\cup\\
&\qquad\qquad \big(\alpha(\langle l_1,2\rangle).\alpha(\langle l_2,2\rangle)\big)\\
&\alpha(\langle l_1.l_2,2\rangle) = \big(\alpha(\langle l_1,1\rangle).\alpha(\langle l_2,2\rangle)\big)\cup\\
&\qquad\qquad \big(\alpha(\langle l_1,2\rangle).\alpha(\langle l_2,1\rangle)\big)\\
&\alpha(\mathtt{exp}(p,e)) = \{p^e\}\\
&\alpha(p.free) = \{q \mid q \geq p\}
\end{aligned}
\qquad
\begin{aligned}
&\alpha(\langle l_1 + l_2, j\rangle) = \alpha(\langle l_1,j\rangle) \cup \alpha(\langle l_2,j\rangle)\\
&\alpha(\langle l_1 \oplus l_2, 1\rangle) = \alpha(\langle l_1,1\rangle) \cup \alpha(\langle l_2,2\rangle)\big)\\
&\alpha(\langle l_1 \oplus l_2, 2\rangle) = \alpha(\langle l_1,2\rangle) \cup \alpha(\langle l_2,1\rangle)\big)\\
&\alpha(c_1.c_2) = \alpha(c_1).\alpha(c_2)\\
&\alpha(\mathtt{exp}(p, k_1N + k_2)) = \{p^{k_1k+k_2} \mid k \geq ||\mathcal{I}||^2\}\\
&\alpha(\mathtt{hps}(u,v)) = \{q \mid v|_q = u\}
\end{aligned}
$$

Figure 2 Definition of the concretization function α.

It it easy to see that, for a 1CU2r instance $\mathcal{I}$ and the corresponding initial state $\mathcal{S}$ (Definition 4.1), $\mathtt{HP}(\mathcal{I}) = \mathtt{HP}(\mathcal{S})$, by Definition 4.2. Hence, to show correctness it suffices to prove that for every rule application, $\mathcal{S} \to_r \mathcal{S}'$ of a rule r on a valid state $\mathcal{S}$, $\mathtt{HP}(\mathcal{S}) \cup \alpha(S) = \mathtt{HP}(\mathcal{S}') \cup \alpha(S')$ holds.

▶ **Lemma 5.8.** *Let $\mathcal{S}$ be a valid state, and let $\mathcal{S} = (R, \Delta, S) \to_r \mathcal{S}' = (R', \Delta', S')$ be a rule r application. Then, $\mathtt{HP}(\mathcal{S}) \cup \alpha(S) = \mathtt{HP}(\mathcal{S}') \cup \alpha(S')$ holds.*

Lemma 4.7 shows that our inference systems makes progress. Lemmas 5.8 and 4.7 imply that, for a given 1CU2r instance $\mathcal{I}$, when our inference system terminates, the obtained set of solutions $S \subset L_s$ satisfies that $\mathtt{HP}(\mathcal{I}) = \bigcup_{l \in S} \alpha(l)$.

Termination follows from the termination argument for the decision version. The termination argument, together with the correctness of our algorithm and the fact that all the rules can be checked for applicability and applied in polynomial time, gives us our main result stated in the following theorem.

▶ **Theorem 5.9.** *The 1CU2r problem can be solved in polynomial time. Moreover, a polynomial size representation of all solutions can be computed in polynomial time. This result holds also when a DAG is used for term representation.*

Finally, we informally argue that our representation satisfies the requirements stated at the beginning of Section 5. First, to extract just one solution, we can expand the definition of the concretization function α (Figure 2) in a depth-first traversal, without backtracking, until we get a concrete position. Second, to get the number m of all solutions, if there is a solution containing the expressions $\mathtt{exp}(p, k_1N + k_2)$ or $p.free$ then m is infinite, otherwise it is at most exponential and it can be efficiently computed using a dynamic programming scheme. Third, all solutions of $\mathcal{I}$ can be obtained in polynomial time w.r.t. m by a simple (depth-first) enumeration of the positions in the solution and expansion of expressions in `BaseCases`.

6 Easy Instances

Recall that our inference rules assume that easy instances of 1CU2r can be solved in polynomial time (Lemma 4.6); and that in fact, a representation for all solutions can also be efficiently computed. We establish those claims here, showing that Lemma 4.6 holds for all cases of Definition 4.5.

The special case where we have one equation that contains F on both sides was solved in polynomial time in a previous paper [8] (Theorem 5.20). That case corresponds, after

abstracting proper subterms of the form $F(r)$ by variables, to the case where $\mathcal{I}$ contains equations of the form $F(u_1) \doteq s, F(u_2) \doteq C[s]$ with $C \neq \bullet$. The key observation is that the hole position of F has to be a prefix or exponentiation of $\mathtt{hp}(C)$. In the problem considered in [8], the terms at the input were given explicitly. Since, we assume the DAG representation for terms, the result from [8] needs to be extended for our setting. However, the proof ideas are the same, and hence the proof is omitted here.

▶ **Lemma 6.1** (cyclic). *Let $\mathcal{I}$ be a 1-CU instance of the form $\mathcal{I} = \mathcal{I}' \cup \{F(u_1) \doteq s, F(u_2) \doteq C[s]\}$, where C is a non-empty context. Then, a polynomial time procedure* `SolveCyclic` *returns a complete representation of* $\mathtt{HP}(\mathcal{I})$ *of polynomial size.*

The expressions in the complete set of solutions of the previous lemma are of the forms $\mathtt{exp}(p, e)$ and $\mathtt{exp}(p, k_1N + k_2)$, where p is a position, e, k_1, k_2 are natural numbers bounded by $||\mathcal{I}||^2$, and N is an integer variable. The expressions $\mathtt{exp}(p, e)$ and $\mathtt{exp}(p, k_1N + k_2)$ stand for the result of raising p to e and $k_1N + k_2$, respectively.

Note that the fact that Lemma 4.6 holds for case (a) of Definition 4.5 follows from Lemma 6.1. The fact that Lemma 4.6 also holds for case (b) of Definition 4.5 follows from the following lemma.

▶ **Lemma 6.2** (Clash). *Let $\mathcal{I}$ be a 1CU instance of the form $\mathcal{I} = \mathcal{I}' \cup \{F(u_1) \doteq f(s_1, \ldots, s_m), F(u_2) \doteq g(t_1, \ldots, t_{m'})\}$, with $f \neq g$. Then, $\mathcal{I}$ can be solved by a polynomial time procedure* `SolveClash` *and* $\mathtt{HP}(\mathcal{I})$ *is either empty or* $\{\lambda\}$.

Proof. The fact that every solution σ must satisfy $F\sigma = \bullet$ directly follows from $f \neq g$. Hence, this particular case reduces to the first-order unification problem $\mathcal{I}' = \mathcal{I}\{F \to \bullet\}$, which can be solved in polynomial time w.r.t. $||\mathcal{I}||$. ◀

Let us now argue that Lemma 4.6 holds for cases (c) and (d) in Definition 4.5. Note that such cases reduce to solving one equation by unifying left hand-sizes and replacing first-order variables by their left hand-sides, respectively. Hence, these two cases follow from the following Lemma. Its proof is included in [9] and hence omitted here.

▶ **Lemma 6.3** (1-eqn). *Let $\mathcal{I}$ be a 1CU2r instance consisting of the single equation $F(s) \doteq t$. Then, a complete representation of* $\mathtt{HP}(\mathcal{I})$ *of polynomial size can be computed in polynomial time.*

The expressions in the complete set of solutions in the previous lemma are of the forms $\mathtt{hps}(u, v)$ and $p.free$ for terms u, v and a position p of polynomial size. These expressions stand for the (possibly exponential and infinite) sets of positions $\{q \in \mathtt{pos}(v) \mid v|_q = u\}$ and $\{p.p' \mid p' \in \mathtt{pos}(\mathcal{F})\}$, respectively.

Finally, we just need to show that Lemma 4.6 also holds for case (e) of Definition 4.5 to complete its proof. Note that such case reduces to solving an instance of the form $\{F(r) \doteq s, F(C[s]) \doteq t\}$. Before we give a polynomial time algorithm for that case, we first prove a particular case of 1CU: solving a single equation of the form $F(C[F(s)]) = t$. Equations of this form have the nice property that the hole position of any context that is a solution for F can not be an extension of a *nonlinear* positions in t. A position p is *nonlinear* in t if there exists another position $q \neq p$ such that $t|_p = t|_q$. We also call $t|_p$ a nonlinear subterm of t. Note that there are only a (linear number of) linear positions in a term. In contrast, there can be exponentially many nonlinear positions in a term. Since the following lemma is already proved in [9], we only state it here.

▶ **Lemma 6.4.** *Let $\mathcal{I}$ be a 1CU instance consisting of one single equation of the form $F(C[F(s)]) \doteq t$ such that F does not occur in t (but F can occur in C). Let $P = \{p \in \mathtt{pos}(t) \mid t|_p = v$ and v is a non-linear subterm of $t\}$. Then, $\forall p \in P : \mathtt{hp}(F\sigma) \neq p$, for every solution σ of $F(C[F(s)]) \doteq t$.*

It follows from the previous Lemma that for the equation $F(C[F(s)]) \doteq t$, we only need to test the (small number of) linear positions as possible hole positions. This implies that, as stated in the following lemma, we can enumerate a *complete* set of unifiers: a set of unifiers is *complete* if any other unifier (for the problem) is an instance of some unifier in the set. Here, a unifier is allowed to instantiate a context variable F in terms of a new context variable F'.

▶ **Lemma 6.5.** *Let $\mathcal{I}$ be a 1CU instance consisting of one single equation of the form $F(C[F(s)]) \doteq t$ such that F does not occur in t. Then, a complete set of unifiers U of $\mathcal{I}$ of polynomial size can be computed in polynomial time. Any substitution σ in U satisfies one of the two conditions below:*

1. *Either $F\sigma = t[\bullet]_p$, with $p \in \mathtt{pos}(t)$,*
2. *Or $\sigma = \{F \mapsto t[F'(\bullet)]_q,\ x \mapsto F'(C[t[F'(s)]_q])\}$, where x does not occur in $F(C[F(s)])$, $t|_q = x$, and F' is a new context variable different from F.*

Proof. We distinguish two cases. First consider solutions σ satisfying that $\mathtt{hp}(F\sigma) \in \mathtt{pos}(t)$. By Lemma 6.4, for each of such solutions, $\mathtt{hp}(F\sigma)$ must be in the set $Q = \{p \in \mathtt{pos}(t) \mid t|_p = v$ and v is a linear subterm of $t\}$. Note that $\{F \to t[\bullet]_p\} \leq \sigma$. Since $|Q|$ is polynomial w.r.t. $|t|$ even in the DAG representation, then U has a polynomial number of solutions of this form. Now consider solutions σ such that $\mathtt{hp}(F\sigma) \notin \mathtt{pos}(t)$. Let $p = p_1.p_2$, where p_1 is the longest prefix of p defined in t. Note that $t|_{p_1}$ must be a variable x linear in t by Lemma 6.4. Moreover, since σ satisfies $x\sigma|_{p_2} = C[F(s)]\sigma$, x does not occur in $C[F(s)]$. Hence, σ is of the form $\{F \mapsto t[D]_{p_1}, x \mapsto DC[t[D(s)]_{p_1}]\}$, for an arbitrary context D such that $\mathtt{hp}(D) = p_2$. Hence, all solutions σ such that $\mathtt{hp}(F\sigma) = q.q'$ and $q.q' \notin \mathtt{pos}(t)$, with $q \in Q$, can be represented by a substitution $\theta = \{F \mapsto t[[F'(\bullet)]]_q, x \mapsto F'(C[t[F'(s)]_q])\}$, where $t|_q = x$, F' is a new context variable different from F, since $\theta \leq \sigma$ holds. ◀

Finally, the fact that Lemma 4.6 holds for case (e) in Definition 4.5 follows from the following lemma, which completes the proof of Lemma 4.6. It is easy to see that the set of solutions of the next lemma can be represented using expressions $\mathtt{hps}(u, v)$ and $p.free$ for terms u, v and a position p of polynomial size, as in Lemma 6.3. Hence, Lemma 4.6 follows from Lemmas 6.1, 6.2, 6.3, and 6.6. Moreover a representation for all solutions consisting of expressions in `BaseCases` can always be computed for all easy instances.

▶ **Lemma 6.6.** *Consider a 1CU2r instance of the form $\mathcal{I} = \{F(r) \doteq s, F(C[s]) \doteq t\}$ Then, a complete representation of $\mathtt{HP}(\mathcal{I})$ of polynomial size can be computed in polynomial time.*

Proof. First note that if either s or t is a constant, the lemma is straightforward. Also, we can whether λ is in $\mathtt{HP}(\mathcal{I})$ by solving the first-order unification problem resulting from applying the substitution $F \mapsto \bullet$. Hence, we can assume that s and t are both not constants and $F\sigma \neq \bullet$, for every solution σ. By Lemma 6.5 we can compute, in polynomial time, a complete set of unifiers $U = \theta_1, \ldots, \theta_k$ of the single equation $F(C[F(r)]) \doteq t$ of polynomial size. Moreover, every substitution $\theta \in U$ satisfies one of the two conditions below:

1. $F\theta = t[\bullet]_p$, with $p \in \mathtt{pos}(t)$, or
2. $\theta = \{F \mapsto t[F'(\bullet)]_q, x \mapsto F'(C[t[F'(r)]_q])\}$, where x does not occur in $F(C[F(r)])$, $t|_q = x$, and F' is a new context variable different from F.

Hence, to obtain a polynomial time algorithm, it is enough to check if some substitution in U can be extended to solve also the equation $F(r) \doteq s$, and thus $\mathcal{I}$. Consider the two cases of a substitution $\theta \in U$.

1. If θ is such that $F\theta = t[\bullet]_p$, then the check can clearly be done in polynomial time, since $F(r)\theta \doteq s\theta$ is a first-order unification instance of polynomial size thanks to the DAG representation.
2. Otherwise, θ is of the form $\{F \mapsto t[F'(\bullet)]_q, x \mapsto F'(C[t[F'(r)]_q])\}$, where $t_q = x$, we distinguish cases depending on whether x occurs in s, and if so, where.
   - **(i)** First assume that x occurs in s at a position p such that either $p < q$ or $p > q$. Since we are looking for solution σ where $\mathtt{hp}(F\sigma) \geq q$, these cases can be rewritten into a form that is covered by Lemma 6.1. Note that since $p \neq q$, C in Lemma 6.1 is nonempty.
   - **(ii)**) Assume that, for some p, $s|_p = x$ and p is disjoint with q. In this case, every solution σ that is an extension of θ satisfies that $|F\sigma| > |F\sigma|_q\sigma| = |x\sigma| = |F(C[F(r)])\sigma| > |F\sigma|$, a contradiction. Hence we know that if x occurs in s at a position disjoint with q there are no solutions that are extensions of θ and thus there is no need to test θ.
   - **(iii)** Consider now the case where $s|_q = x$. In this case, every solution σ that is an extension of θ will satisfy $s\sigma = t\sigma$, and hence, also in this case, there is no need to test θ, since the equation $F(C[s]) \doteq t$, and our assumption that F is not $\bullet$ in any solution, imply $s\sigma \neq t\sigma$ for every solution σ.
   - **(iv)** Finally, consider the case where x does not occur in s. Note that if r contains x then $\mathcal{I}$ has no solution that is an extension of θ, again because of F is not $\bullet$ in any solution. Thus, neither $r\theta$ nor $s\theta$ contain F', $F(r)\theta \doteq s\theta$ reduces to a single equation $F'(r') \doteq s'$ after applying a few first-order decomposition steps, and the lemma follows from Lemma 6.3 in this case. ◀

7 Conclusion

We have shown that the subcase of the one context unification problem of two equations $F(r_1) = s_1, F(r_2) = s_2$ can be solved in polynomial time, including a polynomial size representation of all unifiers. Our procedure led to polynomial time algorithm for several subclasses of the general 1CU problem [9]. We leave it to future work to extend this work to the unrestricted one context unification problem, by extending the techniques presented in this paper and [9]. Another possible future line of research is to investigate the algorithmic properties of fault tolerant unification with other bounds on the number of faults, and whether our algorithm can be extended to obtain a polynomial time algorithm also for the case where STGs are used for term representation.

References

1 F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, New York, NY, USA, 1998.

2 F. Baader and W. Snyder. Unification theory. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, pages 445–532. Elsevier and MIT Press, 2001.

3 H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree Automata Techniques and Applications. Available on: `http://www.grappa.univ-lille3.fr/tata`, 2007. release October, 12th 2007.

4 C. Creus, A. Gascón, and G. Godoy. One-context Unification with STG-Compressed Terms is in NP. In *RTA*, pages 149–164, 2012.

5 S. Schulze Frielinghaus and H. Seidl M. Petter. Inter-procedural Two-variable herbrand Equalities. In *Proc. Symp. Programming, ESOP*, LNCS 9032, pages 457–482, 2015.

6 A. Gascón, G. Godoy, and M. Schmidt-Schauß. Context Matching for Compressed Terms. In *LICS*, pages 93–102, 2008.

7 A. Gascón, G. Godoy, and M. Schmidt-Schauß. Unification and Matching on Compressed Terms. *ACM Trans. Comput. Log.*, 12(4):26, 2011.

8 A. Gascón, G. Godoy, M. Schmidt-Schauß, and A. Tiwari. Context Unification with One Context Variable. *J. Symb. Comput.*, 45(2):173–193, 2010.

9 A. Gascón, M. Schmidt-Schauß, and A. Tiwari. One Context Unification Problems Solvable in Polynomial Time. In *LICS*, 2015. To appear.

10 W. D. Goldfarb. The Undecidability of the Second-Order Unification Problem. *Theor. Comput. Sci.*, 13:225–230, 1981.

11 S. Gulwani and A. Tiwari. Computing procedure summaries for interprocedural analysis. In *ESOP*, pages 253–267, 2007.

12 A. Jeż. Context unification is in PSPACE. In *ICALP*, pages 244–255, 2014.

13 J. Levy, M. Schmidt-Schauß, and M. Villaret. Monadic Second-Order Unification is NP-Complete. In *RTA*, pages 55–69, 2004.

14 J. Levy, M. Schmidt-Schauß, and M. Villaret. Bounded Second-Order Unification is NP-complete. In *RTA*, pages 400–414, 2006.

15 J. Levy, M. Schmidt-Schauß, and M. Villaret. Stratified Context Unification is NP-complete. In *IJCAR*, pages 82–96, 2006.

16 J. Levy, M. Schmidt-Schauß, and M. Villaret. The Complexity of Monadic Second-Order Unification. *SIAM J. Comput.*, 38(3):1113–1140, 2008.

17 J. Levy, M. Schmidt-Schauß, and M. Villaret. On the complexity of Bounded Second-Order Unification and Stratified Context Unification. *Logic Journal (IGPL)*, 19(6):763–789, 2011.

18 M. Lohrey. Algorithmics on SLP-compressed strings: A survey. *Groups Complexity Cryptology*, 4(2):241–299, 2012.

19 M. S. Paterson and M. N. Wegman. Linear unification. *J. of Computer and System Sciences*, 16:158–167, 1978.

20 M. Schmidt-Schauß and K. U. Schulz. Solvability of Context Equations with Two Context Variables is Decidable. *J. Symb. Comput.*, 33(1):77–122, 2002.

Confluence of Layered Rewrite Systems

Jiaxiang Liu[1,2], Jean-Pierre Jouannaud[2], and Mizuhito Ogawa[3]

1 **School of Software, and TNList, Tsinghua University, Beijing, China**
2 **Deducteam, INRIA, and LIX, École Polytechnique, Palaiseau, France**
3 **School of Information Science, JAIST, Ishikawa, Japan**

Abstract

We investigate the new, Turing-complete class of *layered* systems, whose lefthand sides of rules can only be overlapped at a multiset of disjoint or equal positions. Layered systems define a natural notion of rank for terms: the maximal number of non-overlapping redexes along a path from the root to a leaf. Overlappings are allowed in finite or infinite trees. Rules may be non-terminating, non-left-linear, or non-right-linear. Using a novel unification technique, *cyclic unification*, we show that rank non-increasing layered systems are confluent provided their cyclic critical pairs have cyclic-joinable decreasing diagrams.

1998 ACM Subject Classification F.4.2 [Logic and computation] Rewriting

Keywords and phrases Layers, confluence, decreasing diagrams, critical pairs, cyclic unification

Digital Object Identifier 10.4230/LIPIcs.CSL.2015.423

1 Introduction

Confluence of terminating systems is well understood: it can be reduced to the joinability of local peaks by Newman's lemma, and to that of critical ones, obtained by unifying lefthand sides of rules at subterms, by Knuth-Bendix-Huet's lemma. Confluence can thus be decided by inspecting all critical pairs, see for example [5].

Many efforts notwithstanding [23, 12, 27, 20, 22, 19, 29, 10, 14, 24, 11, 30, 15, 25, 18, 1], confluence of non-terminating systems is far from being understood in terms of critical pairs. Only recently did this question make important progress with van Oostrom's complete method for checking confluence based on decreasing diagrams, a generalization of joinability [28, 29]. In particular, while Huet's result stated that linear systems are confluent provided their critical pairs are strongly confluent [12], Felgenhauer showed that right-linearity could be removed provided parallel critical pairs have decreasing diagrams [8]. Knuth-Bendix's and Felgenhauer's theorems can join forces in presence of both terminating and non-terminating rules [17].

We show here that rank non-increasing layered systems are confluent provided their critical pairs have decreasing diagrams. Our confluence result for non-terminating non-linear systems by critical pair analysis is the first we know of. Further, our result holds in case critical pairs become infinite, solving a long standing problem raised in [12]. Prior solutions to the problem existed under different assumptions that could be easily challenged [27, 10, 15].

Our results use a simplified version of sub-rewriting introduced in [17], and a simple, but essential revisitation of unification in case overlaps generate occur-check equations: *cyclic unification* is based on a new, important notion of cyclic unifiers, which enjoy all good properties of unifiers over finite trees such as existence of most general cyclic unifiers, and can therefore represent solutions of occur-check equations by simple rewriting means.

24th EACSL Annual Conference on Computer Science Logic (CSL 2015).
Editor: Stephan Kreutzer; pp. 423–440

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Terms are introduced in Section 2, labelled rewriting and decreasing diagrams in Section 3, sub-rewriting in Section 4, cyclic unification in Section 5 and layered systems in Section 6 where our main result is developed, before concluding in Section 7.

2 Terms, substitutions, and rewriting

Given a *signature* $\mathcal{F}$ of *function symbols* and a denumerable set $\mathcal{X}$ of *variables*, $\mathcal{T}(\mathcal{F},\mathcal{X})$ denotes the set of finite or infinite rational *terms* built up from $\mathcal{F}$ and $\mathcal{X}$. We reserve letters x,y,z for variables, f,g,h for function symbols, and s,t,u,v,w for terms. Terms are recognized by top-down tree automata in which some ω-states, and only those, are possibly traversed infinitely many times. Terms are identified with labelled trees. See [4] for details.

Positions are finite strings of positive integers. We use o,p,q for arbitrary positions, the empty string Λ for the root position, and "$\cdot$" for concatenation of positions or sets thereof. We use $\mathcal{FP}os(t)$ for the (possibly infinite) set of non-variable positions of t, $t(p)$ for the function symbol at position p in t, $t|_p$ for the *subterm* of t at position p, and $t[u]_p$ for the result of replacing $t|_p$ with u at position p in t. We may omit the position p, writing $t[u]$ for simplicity and calling $t[\cdot]$ a *context*. We use $\geq$ for the partial prefix order on positions (further from the root is bigger), $p\#q$ for incomparable positions p,q, called *disjoint*. The order on positions is extended to finite sets as follows: $P\geq Q$ (resp. $P>Q$) if $(\forall p\in P)(\exists q\in max(Q))\, p\geq q$ (resp. $p>q$), where $max(P)$ is the set of maximal positions in P. We use p for the set $\{p\}$.

We use $\mathcal{V}ar(t_1,\ldots,t_n)$ for the set of variables occurring in $\{t_i\}_i$. A term t is *ground* if $\mathcal{V}ar(t)=\emptyset$, *linear* if no variable occurs more than once in t. Given a term t, we denote by $\bar{t}$ any linear term obtained by renaming, for each variable $x\in\mathcal{V}ar(t)$, the occurrences of x at positions $\{p_i\}_i$ in t by *linearized variable* x^{k_i} such that $i\neq j$ implies $x^{k_i}\neq x^{k_j}$. Note that $\mathcal{V}ar(\bar{s})\cap\mathcal{V}ar(\bar{t})=\emptyset$ iff $\mathcal{V}ar(s)\cap\mathcal{V}ar(t)=\emptyset$. Identifying x^{k_0} with x, $\bar{t}=t$ for a linear term t.

A *substitution* σ is an endomorphism from terms to terms defined by its value on its *domain* $\mathcal{D}om(\sigma):=\{x : \sigma(x)\neq x\}$. Its *range* is $\mathcal{R}an(\sigma):=\bigcup_{x\in\mathcal{D}om(\sigma)}\mathcal{V}ar(x\sigma)$. We use $\sigma_{|V}$ for the restriction of σ to $V\subseteq\mathcal{D}om(\sigma)$, and $\sigma_{|\neg X}$ for the restriction of σ to $\mathcal{D}om(\sigma)\setminus X$. The substitution σ is said to be *finite* (resp., a *variable substitution*) if for each $x\in\mathcal{D}om(\sigma)$, $\sigma(x)$ is a finite term (resp., a variable). Variable substitutions are called *renamings* when also bijective. A substitution γ is *ground* if for each $x\in\mathcal{X}$, $\gamma(x)$ is ground. We use Greek letters for substitutions and postfix notation for their application.

The strict *subsumption order* $\gtrdot$ on finite terms (resp. substitutions) associated with the quasi-order $s\gtrsim t$ (resp. $\sigma\gtrsim\tau$) iff $s=t\theta$ (resp. $\sigma=\tau\theta$) for some substitution θ, is well-founded.

A *rewrite rule* is a pair of finite terms, written $l\to r$, whose *lefthand side* l is not a variable and whose *righthand side* r satisfies $\mathcal{V}ar(r)\subseteq\mathcal{V}ar(l)$. A *rewrite system* R is a set of rewrite rules. A rewrite system R is *left-linear* (resp. *right-linear*, *linear*) if for every rule $l\to r\in R$, l is a linear term (resp. r is a linear term, l and r are linear terms). Given a rewrite system R, a term u *rewrites* to a term v *at* a position p, written $u\xrightarrow[R]{p}v$, if $u|_p=l\sigma$ and $v=u[r\sigma]_p$ for some rule $l\to r\in R$ and substitution σ. The term $l\sigma$ is a *redex* and $r\sigma$ its *reduct*. We may omit R as well as p, and also replace the former by the rule which is used and the latter by a property it satisfies, writing for example $u\xrightarrow[l\to r]{>P}v$. Rewriting *terminates* if there exists no infinite rewriting sequence issuing from some term. Rewriting is sometimes called *plain rewriting*.

Consider a *local peak* made of two rewrites issuing from the same term u, say $u\to^{p}_{l\to r}v$ and $u\to^{q}_{g\to d}w$. Following Huet [12], we distinguish three cases:

$p\#q$ (disjoint case), $q>p\cdot\mathcal{FP}os(l)$ (ancestor case), and $q\in p\cdot\mathcal{FP}os(l)$ (critical case). Given two, possibly different rules $l\to r, g\to d$ and a position $p\in\mathcal{FP}os(l)$ such that

$\mathcal{V}ar(l) \cap \mathcal{V}ar(g) = \emptyset$ and σ is a most general unifier of the equation $l|_p = g$, then $l\sigma$ is the *overlap* and $\langle r\sigma, l\sigma[d\sigma]_p\rangle$ the *critical pair* of $g \to d$ on $l \to r$ at p.

Rewriting extends naturally to lists of terms of the same length, hence to substitutions of the same domain. See [5, 26] for surveys.

3 Labelled rewriting and decreasing diagrams

Our goal is to reduce confluence of a non-terminating rewrite system R to that of finitely many critical pairs. Huet's analysis of linear non-terminating systems was based on Hindley's lemma, stating that a non-terminating relation is confluent provided its local peaks are joinable in at most one step from each side [12]. The more general analyses needed here have been made possible by van Oostrom's notion of decreasing diagrams for labelled relations.

▶ **Definition 1.** A *labelled rewrite relation* is a pair made of a rewrite relation $\to$ and a mapping from rewrite steps to a set of labels $\mathcal{L}$ equipped with a partial quasi-order $\trianglerighteq$ whose strict part $\triangleright$ is well-founded. We write $u \xrightarrow[R]{p,m} v$ for a rewrite step from u to v at position p with label m and rewrite system R. Indexes p, m, R may be omitted. We also write $\alpha \triangleright l$ (resp. $l \triangleright \alpha$) if $m \triangleright l$ (resp. $l \triangleright m$) for all m in a multiset α.

Given an arbitrary labelled rewrite step $\to^l$, we denote its projection on terms by $\to$, its inverse by ${}^l\!\leftarrow$, its reflexive closure by $\Rightarrow^l$, its symmetric closure by $\overset{l}{\leftrightarrow}$, its reflexive, transitive closure by $\twoheadrightarrow^\alpha$ for some word α on the alphabet of labels, and its reflexive, symmetric, transitive closure, called *conversion*, by $\overset{\alpha}{\twoheadleftarrow\!\!\twoheadrightarrow}$. We may consider the word α as a multiset.

The triple v, u, w is said to be a *local peak* if $v \,{}^l\!\leftarrow u \to^m w$, a *peak* if $v \,{}^\alpha\!\twoheadleftarrow u \twoheadrightarrow^\beta w$, a *joinability diagram* if $v \twoheadrightarrow^\alpha u \,{}^\beta\!\twoheadleftarrow w$. The local peak $v \,{}^{p,m}_{l\to r}\!\leftarrow u \to^{q,n}_{g\to d} w$ is a *disjoint, critical, ancestor* peak if $p \# q$, $q \in p\cdot\mathcal{FP}os(l)$, $q > p\cdot\mathcal{FP}os(l)$, respectively. The pair v, w is *convertible* if $v \overset{\alpha}{\twoheadleftarrow\!\!\twoheadrightarrow} w$, *divergent* if $v \,{}^\alpha\!\twoheadleftarrow u \twoheadrightarrow^\beta w$ for some u, and *joinable* if $v \twoheadrightarrow^\alpha t \,{}^\beta\!\twoheadleftarrow w$ for some t. The relation $\to$ is *locally confluent* (resp. *confluent*, *Church-Rosser*) if every local peak (resp. divergent pair, convertible pair) is joinable.

Given a labelled rewrite relation $\to^l$ on terms, we consider specific conversions associated with a given local peak called *local diagrams* and recall the important subclass of van Oostrom's decreasing diagrams and their main property: a relation all whose local diagrams are decreasing enjoys the Church-Rosser property, hence confluence. Decreasing diagrams were introduced in [28], where it is shown that they imply confluence, and further developed in [29]. The first version suffices for our needs.

▶ **Definition 2** (Local diagrams)**.** A *local diagram* D is a pair made of a *local peak* $D_{peak} = v \leftarrow u \to w$ and a *conversion* $D_{conv} = v \overset{}{\twoheadleftarrow\!\!\twoheadrightarrow} w$. We call *diagram rewriting* the rewrite relation $\Rightarrow_{\mathcal{D}}$ on conversions associated with a set $\mathcal{D}$ of local diagrams, in which a local peak is replaced by one of its associated conversions:

$$P\ D_{peak}\ Q \underset{\mathcal{D}}{\Rightarrow} P\ D_{conv}\ Q \text{ for some } D \in \mathcal{D}$$

▶ **Definition 3** (Decreasing diagrams [28])**.** A local diagram D with peak $v \,{}^l\!\leftarrow u \to^m w$ is *decreasing* if its conversion $D_{conv} = v \overset{\alpha}{\twoheadrightarrow} s \overset{m}{\Rightarrow} s' \overset{\delta}{\twoheadrightarrow} \overset{\delta'}{\twoheadleftarrow} t' \,{}^l\!\Leftarrow t \,{}^\beta\!\twoheadleftarrow w$ satisfies the following *decreasingness condition*: labels in α (resp. β) are strictly smaller than l (resp. m), and labels in δ, δ' are strictly smaller than l or m. The rewrites $s \Rightarrow^m s'$ and $t' \,{}^l\!\Leftarrow t$ are called the *facing steps* of the diagram.

▶ **Theorem 4** (14])**.** *The relation $\Rightarrow_{\mathcal{D}}$ terminates for any set $\mathcal{D}$ of decreasing diagrams.*

▶ **Corollary 5.** *Assume that $T \subseteq \mathcal{T}(\mathcal{F},\mathcal{X})$ and $\mathcal{D}$ is a set of decreasing diagrams in T such that the set of T-conversions is closed under $\Rightarrow_{\mathcal{D}}$. Then, the restriction of $\to$ to T is Church-Rosser if every local peak in T has a decreasing diagram in $\mathcal{D}$.*

This simple corollary of Theorem 4 is a reformulation of van Oostrom's decreasing diagram theorem which is convenient for our purpose.

4 Sub-rewriting

Consider the following famous system inspired by an abstract example of Newman, algebraized by Klop and publicized by Huet [12], NKH $= \{f(x,x) \to a,\ f(x,c(x)) \to b,\ g \to c(g)\}$. NKH is overlap-free, hence locally confluent by Huet's lemma [12]. However, it enjoys non-joinable non-local peaks such as $a \leftarrow f(g,g) \to f(g,c(g)) \to b$.

The main difficulty with NKH is that non-joinable peaks are non-local. To restore the usual situation for which the confluence of a relation can be characterized by the joinability of its local peaks, we need another rewrite relation whose local peaks capture the non-confluence of NKH as well as the confluence of its confluent variations. A major insight of [17] is that this can be achieved by the *sub-rewriting* relation, that allows us to rewrite $f(g,c(g))$ in one step to either a or b, therefore exhibiting the pair $\langle a,b\rangle$ as a sub-rewriting critical pair. Sub-rewriting is made of a preparatory *equalization* phase in which the variable instances of the lefthand side l of some rule $l \to r$ are joined, taking place before the rule is applied in the *firing phase*. In [17], sub-rewriting required a signature split to define layers in terms, the preparatory phase taking place in the lower layers. No a-priori layering is needed here:

▶ **Definition 6** (Sub-rewriting)**.** A term u *sub-rewrites* to a term v at $p \in \mathcal{P}os(u)$ for some rule $l \to r \in R$, written $u \to^{p}_{R_R} v$, if $u \twoheadrightarrow_R^{(>p\cdot\mathcal{F}\mathcal{P}os(l))} u[l\theta]_p \to^{p}_R u[r\theta]_p = v$ for some substitution θ. The term $u|_p$ is called a *sub-rewriting redex*.

This definition of sub-rewriting allows *arbitrary rewriting* below the lefthand side of the rule until a redex is obtained. This is the major idea of sub-rewriting, ensuring that $R \subseteq R_R \subseteq R^*$. A simple, important property of a sub-rewriting redex is that it is an instance of a linearized lefthand side of rule:

▶ **Lemma 7** (Sub-rewriting redex)**.** *Assume u sub-rewrites to $u[r\sigma]_p$ with $l \to r$ at position p. Then $u|_p = \bar{l}\theta$ for some θ s.t. $(\forall x \in \mathcal{V}ar(l))\,(\forall p_i \in \mathcal{P}os(l)\ s.t.\ l(p_i){=}x)\,\theta(x^{p_i}) \twoheadrightarrow_R \sigma(x)$.*
We say that σ is an equalizer *of l, and the rewrite steps from $\bar{l}\theta$ to $l\sigma$ are an* equalization*.*

Sub-rewriting differs from rewriting modulo by being directional. It differs from Klop's higher-order rewriting modulo developments [26] used by Okui for first-order computations [22], in that the preparatory phase uses arbitrary rewriting. Having non-left-linear rules with critical pairs at subterms seems incompatible with using developments. Sub-rewriting differs as well from relative rewriting [11] in that the preparatory phase must take place below variables. The latter condition is essential to obtain plain critical pairs based on plain unification.

Assuming that local sub-rewriting peaks characterize the confluence of NKH, we need to compute the corresponding critical pairs. Unifying the lefthand sides $f(x,x)$ and $f(y,c(y))$ results in the conjunction $x = y \wedge y = c(y)$ containing the *occur-check equation* $y = c(y)$, which prevents unification from succeeding on finite trees but allows it to succeed on infinite rational trees: the critical peak has therefore an infinite overlap $f(c^\omega,c^\omega)$ and a finite critical pair $\langle a,b\rangle$. At the level of infinite trees, we then have an infinite local rewriting peak

Remove	$s = s \wedge P$	$\rightarrow$	P	
Decomp	$f(\vec{s}) = f(\vec{t}) \wedge P$	$\rightarrow$	$\vec{s} = \vec{t} \wedge P$	
Conflict	$f(\vec{s}) = g(\vec{t}) \wedge P$	$\rightarrow$	$\bot$	**if** $f \neq g$
Choose	$y = x \wedge P$	$\rightarrow$	$x = y \wedge P$	**if** $x \notin Var(P)$, $y \in \mathcal{V}ar(P)$
Coalesce	$x = y \wedge P$	$\rightarrow$	$x = y \wedge P\{x \mapsto y\}$	**if** $x, y \in Var(P)$, $x \neq y$
Swap	$u = x \wedge P$	$\rightarrow$	$x = u \wedge P$	**if** $u \notin \mathcal{X}$
Merge	$x = s \wedge x = t \wedge P$	$\rightarrow$	$x = s \wedge s = t \wedge P$	**if** $x \in \mathcal{X}$, $0 < \|s\| \leq \|t\|$
Replace	$x = s \wedge P$	$\rightarrow$	$x = s \wedge P\{x \mapsto s\}$	**if** $x \in Var(P)$, $x \notin \mathcal{V}ar(s)$, $s \notin \mathcal{X}$
Merep	$y = x \wedge x = s \wedge P$	$\rightarrow$	$y = s \wedge x = s \wedge P$	**if** $x \in Var(s)$, $s \notin \mathcal{X}, y \notin \mathcal{V}ar(s, P)$
				and no other rule applies

Figure 1 Unification Rules.

$a \leftarrow f(c^\omega, c^\omega) = f(c^\omega, c(c^\omega)) \rightarrow b$, the properties of infinite trees making the sub-rewriting preparatory phase useless. Sub-rewriting therefore captures on finite trees some properties of rewriting on infinite trees, here the existence of a local peak. Computing the critical pairs of the sub-rewriting relation is therefore related to unification over finite trees resulting possibly in solutions over infinite rational trees. In the next section, we develop a novel view of unification that will allow us to capture both finite and infinite overlaps by finite means.

5 Cyclic unification

This section is adapted from [3, 5, 13] by treating finite and infinite unifiers uniformly: equality of terms is interpreted over the set of infinite rational terms *when needed*.

An *equation* is an oriented pair of finite terms, written $u = v$. A *unification problem* P is a (finite) conjunction $\wedge_i u_i = v_i$ of equations, sometimes seen as a multiset of pairs written $\vec{u} = \vec{v}$. A *unifier* (resp. a *solution*) of P is a substitution (resp., a ground substitution) θ such that $(\forall i)\, u_i\theta = v_i\theta$. A unifier describes a generally infinite set of solutions via its ground instances. A major usual assumption, ensuring that solutions exist when unifiers do, is that the set $\mathcal{T}(\mathcal{F})$ of ground terms is non-empty. A unification problem P has a *most general finite unifier* $mgu(P)$, whenever a finite solution exists, which is minimal with respect to subsumption and unique up to variable renaming. Computing $mgu(P)$ can be done by the unifier-preserving transformations of Figure 1, starting with P until a *solved form* is obtained, $\bot$ denoting the absence of solution, whether finite or infinite. Our notion of solved form therefore allows for infinite unifiers (and solutions) as well as finite ones:

▶ **Definition 8.** *Solved forms* of a unification problem P different from $\bot$ are unification problems $S = \vec{x} = \vec{u} \wedge \vec{y} = \vec{v}$ such that

(i) $\mathcal{P} = \mathcal{V}ar(P) \setminus (\vec{x} \cup \vec{y})$ is the set of *parameters* of S;
(ii) variables in $\vec{x} \cup \vec{y}$ (i.e. variables at lefthand sides of equations) are all distinct;
(iii) $(\forall\, x = u \in \vec{x} = \vec{u})$, $\mathcal{V}ar(u) \subseteq \mathcal{P}$;
(iv) $(\forall\, y = v \in \vec{y} = \vec{v})$, $\mathcal{V}ar(v) \subseteq \mathcal{P} \cup \vec{y}$, $\mathcal{V}ar(v) \cap \vec{y} \neq \emptyset$ and $v \notin \mathcal{X}$.

Equations $y = v \in \vec{y} = \vec{v}$ are called *cyclic* (or *occur-check*, the vocabulary originating from [3] used so far), $\vec{x}$ is the set of *finite* variables, and $\vec{y}$ is the set of (infinite) *cyclic* (or *occur-check*) variables. A solved form is a set of equations since $\vec{x} \cup \vec{y}$ is itself a set and an equation $x = y$ between variables can only relate a finite variable x with a parameter y.

▶ **Example 9** (NKH). $f(x,x) = f(y,c(y)) \to_{\mathbf{Decomp}} x = y \wedge x = c(y) \to_{\mathbf{Coalesce}} x = y \wedge y = c(y) \to_{\mathbf{Merep}} x = c(y) \wedge y = c(y)$. Alternatively, $f(x,x) = f(y,c(y)) \to_{\mathbf{Decomp}} x = y \wedge x = c(y) \to_{\mathbf{Replace}} c(y) = y \wedge x = c(y) \to_{\mathbf{Swap}} y = c(y) \wedge x = c(y)$.

Choose and **Swap** originate from [3]. **Replace** and **Coalesce** ensure that finite variables (but parameters) do not occur in equations constraining the infinite ones. **Merep** is a sort of combination of **Merge** and **Replace** ensuring condition $v \notin \mathcal{X}$ in Definition 8, item (iv). Unification over finite trees has another failure rule, called **Occur-check**, fired in presence of cyclic equations.

▶ **Theorem 10.** *Given an input unification problem P, the unification rules terminate, fail if the input has no solution, and return a solved form $S = \vec{x} = \vec{u} \wedge \vec{y} = \vec{v}$ otherwise.*

Proof. Termination, characterization of solved forms, soundness, are all adapted from [13].

Termination. The quadruple $\langle nu, |P|, nvre, nvle\rangle$ is used to interpret a unification problem P, where

- nu is the number of unsolved variables (0 for $\bot$), where a variable x is *solved* in $x = s \wedge P'$ if $x \notin \mathcal{V}ar(s, P')$;
- $|P|$ is the multiset ($\emptyset$ for $\bot$) of natural numbers $\{max(|s|,|t|) : s = t \in P\}$;
- $nvle$ (resp. $nvre$) is the number of equations in P whose lefthand (resp., righthand) side is a variable and the other side is not.

Remove, **Decomp** and **Conflict** decrease $|P|$ without increasing nu. **Choose** and **Coalesce** both decrease nu. **Swap** decreases $nvre$ without increasing nu and $|P|$. **Merge** decreases $nvle$ without increasing nu, $|P|$ and $nvre$. **Replace** decreases nu. Now, when **Merep** applies, no other rule can apply, and we can check that no rules can apply either after **Merep** (except another possible application of **Merep**). This can happen only finitely many times, by simply reasoning on the number of equations whose both sides are variables.

Solved form. We show by contradiction that the output P, which is in normal form with respect to the unification rules, is a solved form in case **Conflict** never applies. First, P must be a conjunction of equations $x = s$, since otherwise **Decomp** or **Swap** would apply. Let $\mathcal{P} = \mathcal{V}ar(P) \setminus (\vec{x} \cup \vec{y})$.

- Condition (i) is a definition.
- Condition (ii). Let $P = x = s \wedge x = t \wedge P'$. Either s or t is a variable, since otherwise **Merge** would apply. Assume without loss of generality that $s \in \mathcal{X}$, call it y. If $x = y$, **Remove** applies. If $y \notin \mathcal{V}ar(t, P')$, then **Choose** applies. Otherwise, **Coalesce** applies. Hence $\vec{x}, \vec{y}$ are all different *sets*, and P is therefore itself a set.
  Let now $\vec{x} = \vec{u}$ be a maximal (with respect to inclusion) set of equations in P such that $\mathcal{V}ar(\vec{u}) \subseteq \mathcal{P}$, and $\vec{y} = \vec{v}$ be the remaining set of equations.
- Condition (iii). It is ensured by the definition of $\vec{x} = \vec{u}$.
- Condition (iv). Let $y = v \in \vec{y} = \vec{v}$.
  Let now $x = u \in \vec{x} = \vec{u}$, hence $x \notin \mathcal{V}ar(u)$. Assume $x \in \mathcal{V}ar(v)$. If $u \notin \mathcal{X}$, then **Replace** applies. Otherwise, if u has no other occurrence in P, then **Choose** applies, else **Coalesce** applies. Therefore $\mathcal{V}ar(v) \cap \vec{x} = \emptyset$ by contradiction.
  Assume $\mathcal{V}ar(v) \cap \vec{y} = \emptyset$. Then $\mathcal{V}ar(v) \subseteq \mathcal{P}$, which contradicts the maximality of $\vec{x} = \vec{u}$.
  We are left to show that v is not a variable. If it were, then $v \in \vec{y}$. First, $v \neq y$, otherwise **Remove** applies. Let $P = (y = v) \wedge P'$ with $v \in \vec{y} \setminus \{y\}$. Let $v = z$, there must exist $(z = w) \in P'$ for some w, otherwise $z \in \mathcal{P}$. Hence $P' = (z = w) \wedge P''$. Now, $y \notin \mathcal{V}ar(w, P'')$, otherwise **Coalesce** applies. Then we show $z \in \mathcal{V}ar(w)$: firstly, $w \neq z$,

otherwise **Remove** applies; secondly, w is not a variable, otherwise $w \notin \mathcal{V}ar(y, P'')$ lets **Choose** apply, while $w \in \mathcal{V}ar(y, P'')$ makes **Coalesce** available; then if $z \notin \mathcal{V}ar(w)$, **Replace** applies. Thus $z \in \mathcal{V}ar(w)$, allowing **Merep**, which contradicts that we have indeed a solved form.

Soundness. The set of solutions is an invariant of the unification rules. This is trivial for all rules but **Coalesce**, **Merge**, **Replace**, **Merep**, for which it follows from the fact that substitutions are homomorphisms and equality is a congruence. ◀

The solved form is a *tree solved form* if $\vec{y} = \emptyset$, and otherwise an Ω *solved form* whose solutions are infinite substitutions taking their values in the set of infinite (rational) terms. We shall now develop our notion of *cyclic unifier* capturing both solved forms by describing the infinite unifiers of a problem P as a pair of a finite unifier σ and a set of cyclic equations E constraining those variables that require infinite solutions. In case $E = \emptyset$, then P is a tree solved form and $\sigma = mgu(P)$. To avoid manipulating infinite unifiers when $E \neq \emptyset$, we shall work with the cyclic equations themselves considered as a ground rewrite system.

▶ **Definition 11** ([21])**.** Given a set of equations E, we denote by $=_E^{cc}$ the equational theory in which the variables in $\mathcal{V}ar(E)$ are treated as constants, also called *congruence closure* E.

We are interested in the congruence closure defined by cyclic equations, seen here as a set R of ground rewrite rules. We may sometimes consider R as a set of equations, to be either solved or used as axioms, depending on context.

▶ **Definition 12.** A *cyclic rewrite system* is a set of rules $R = \{\vec{y} \to \vec{v}\}$ such that the unification problem $\vec{y} = \vec{v}$ is its own solved form with $\vec{y}$ as the set of infinite cyclic variables. Variables in R are treated as constants.

▶ **Lemma 13.** *A cyclic rewrite system R is ground and critical pair free, hence Church-Rosser.*

We now introduce our definition of cyclic unifiers and solutions:

▶ **Definition 14.** A *cyclic unifier* of a unification problem P is a pair $\langle \eta, R\rangle$ made of a substitution η and a cyclic rewrite system $R = \{\vec{y} \to \vec{v}\}$, satisfying:

(i)) $\mathcal{D}om(\eta) \subseteq \mathcal{V}ar(P) \setminus \vec{y}$, $\mathcal{R}an(\eta) \cap \vec{y} = \emptyset$, and $\mathcal{R}an(\eta) \cap \mathcal{D}om(\eta) = \emptyset$;

(ii) P and $P \wedge R$ have identical sets of solutions ; and

(iii) **(a)** $(\forall u = v \in P)\, u\eta =_{R\eta}^{cc} v\eta$, or equivalently by Lemma 13,

(b) $(\forall u = v \in P)\, u\eta \twoheadrightarrow_{R\eta} {}_{R\eta}\!\!\twoheadleftarrow v\eta$.

A *cyclic solution* of P is a pair $\langle \eta\rho, R\rangle$ made of a cyclic unifier $\langle \eta, R\rangle$ of P and an additional substitution ρ.

We shall use (iii)(a) or (iii)(b) indifferently, depending on our needs, by referring to (iii).

The idea of cyclic unifiers is that the need for infinite values for some variables is encoded via the use of the cyclic rewrite system R, which allows us to solve the various occur-check equations generated when unifying P. Finite variables are instantiated by the finite substitution η, which ensures that cyclic unification reduces to finite unification in the absence of infinite variables. The technical restrictions on $\mathcal{D}om(\eta)$ and $\mathcal{R}an(\eta)$ aim at making η idempotent. In (iii), parameters occurring in R are instantiated by η before rewriting takes place: cyclic unification is nothing but rigid unification modulo the cyclic equations in R [9]. Instantiation of the infinite variables $\vec{y}$ is delegated to cyclic solutions via the additional substitution ρ which may also instantiate the variables introduced by η.

▶ **Example 15.** Consider the equation $f(x,z,z) = f(a,y,c(y))$. A cyclic unifier is $\langle\{x \mapsto a\}, \{y \to c(z), z \to c(z)\}\rangle$, and a cyclic solution is $\langle\{x \mapsto a, y \mapsto a, z \mapsto c(a)\}, \{y \to c(z), z \to c(z)\}\rangle$, which is clearly an instance of the former by the substitution $\{y \mapsto a, z \mapsto c(a)\}$. For the former, $f(a,z,z) =^{cc}_{\{y=c(z),z=c(z)\}} f(a,y,c(y))$. Another cyclic unifier is $\langle\{x \mapsto a\}, \{z \to c(y), y \to c(y)\}\rangle$, for which $f(a,z,z) =^{cc}_{\{z=c(y),y=c(y)\}} f(a,y,c(y))$.

The set of cyclic unifiers of a problem P is closed under substitution instance, provided the variable conditions on its substitution part are met, as is the set of its unifiers. Cyclic unifiers have indeed many interesting properties similar to those of finite unifiers, of which we are going to investigate only a few which are relevant to the confluence of layered systems.

We now focus our attention on specific cyclic unifiers sharing a same cyclic rewrite system.

▶ **Definition 16.** Given a unification problem P with solved form $S = \vec{x} = \vec{u} \wedge \vec{y} = \vec{v}$, let

- its set of parameters $\mathcal{P} = \mathcal{V}ar(P) \setminus (\vec{x} \cup \vec{y})$,
- its cyclic rewrite system $R_S = \{\vec{y} \to \vec{v}\}$ and *canonical* substitution $\eta_S = \{\vec{x} \mapsto \vec{u}\}$,
- its S-based cyclic unifiers $\langle \eta, R_S\rangle$, among which $\langle \eta_S, R_S\rangle$ is said to be *canonical*.

We now show a major property of S-based cyclic unifiers, true for any solved form S:

▶ **Lemma 17.** *Given a unification problem with solved form S, the set of S-based cyclic unifiers is preserved by the unification rules.*

Proof. The result is straightforward for **Remove**, **Choose**, and **Swap**. It is true for **Decomp** and **Conflict** since, using formulation (iii.b) of Definition 14, the rules in $R\eta$ cannot apply at the root of $\mathcal{F}$-headed terms. Next comes **Coalesce**. We need to prove that $\langle \eta, R\rangle$ is a cyclic unifier for $x = y \wedge P$ iff it is one for $x = y \wedge P\{x \mapsto y\}$. Let $u = v \in P$. For the only if case, we have $u\{x \mapsto y\}\eta = u\eta\{x\eta \mapsto y\eta\} =^{cc}_{R\eta} u\eta =^{cc}_{R\eta} v\eta =^{cc}_{R\eta} v\eta\{x\eta \mapsto y\eta\} = v\{x \mapsto y\}\eta$. The if case is similar. **Replace** is similar to **Coalesce**. Consider now **Merge** (**Merep** is similar). Showing that $\langle \eta, R\rangle$ is a cyclic unifier for $x = s \wedge x = t \wedge P$ iff it is one for $x = s \wedge s = t \wedge P$ is routine by using transitivity of the congruence closure $=^{cc}_{R\eta}$. ◀

We can now conclude:

▶ **Theorem 18.** *Given a unification problem P with solved form $S = \vec{x} = \vec{u} \wedge \vec{y} = \vec{v}$, the canonical S-based cyclic unifier is most general among the set of S-based cyclic unifiers of P.*

Proof. Let $\langle \eta, R_S\rangle$ be a cyclic unifier of P based on S.

Let $x = u \in \vec{x} = \vec{u}$. By definition of cyclic unification, $x\eta \twoheadrightarrow_{R_S\eta}{}_{R_S\eta}\twoheadleftarrow u\eta$. By definition of a solved form and cyclic unifiers, we have: $\mathcal{V}ar(x\eta, u\eta) \subseteq (\vec{x} \cup \mathcal{P} \cup \mathcal{R}an(\eta))$, $(\vec{x} \cup \mathcal{P}) \cap \vec{y} = \emptyset$, $\mathcal{R}an(\eta) \cap \vec{y} = \emptyset$, and $\vec{y} \cap \mathcal{D}om(\eta) = \emptyset$. Therefore, $x\eta$ and $u\eta$ are irreducible by $R_S\eta$. Hence $x\eta = u\eta$. Since $x\eta_S = u$, it follows that $x\eta = u\eta = (x\eta_S)\eta = x(\eta_S\eta)$.

Let now $z \in \mathcal{V}ar(P) \setminus \vec{x}$. Since $z \notin \mathcal{D}om(\eta_S)$, then $\eta(z) = z\eta = (z\eta_S)\eta = z(\eta_S\eta)$.

Therefore, $\eta = \eta_S\eta$ and we are done. ◀

This result, which suffices for our needs, is easily lifted to cyclic solutions, as they are instances of a cyclic unifier. We can further prove that η_S is more general than any S'-based cyclic unifiers, for any solved form S' of P. This is where our conditions on $\mathcal{R}an(\eta)$ become important. We conjecture that it is most general among the set of all cyclic unifiers.

6 Layered systems

NKH is non-confluent, but can be easily made confluent by adding the rule $a \to b$ (giving NKH^1), or removing the rule $g \to c(g)$ (giving NKH^2). It can be made non-right-ground by making the symbols a, b unary (using $a(c(x))$ and $b(c(x))$ in the righthand sides of rules, giving NKH^3), or even non-right-linear by making them binary (giving NKH^4). There are classes of systems containing NKH for which it is possible to conclude its non-confluence. The following classes succeed for NKH^1: simple-right-linear [27], strongly depth-preserving [10], and relatively terminating [15]. As for NKH^3, it is neither simple-right-linear nor strongly depth-preserving: only [15] can cover it. When it comes to NKH^4, relative termination becomes hard to satisfy in presence of non-right-linearity [15].

Our goal is to define a robust, Turing-complete class of rewrite systems capturing NKH and its variations, for which confluence can be analyzed in terms of critical diagrams.

▶ **Definition 19.** A rewrite system R is *layered* iff it satisfies the *disjointness* assumption (DLO) that *linearized* overlaps of some lefthand sides of rules upon a given lefthand side l can only take place at a multiset of disjoint or equal positions of $\mathcal{FP}os(l)$:

$$\begin{array}{rcl}
\text{(DLO)} & := & (\forall l \to r \in R)\,(\forall p \in \mathcal{FP}os(l))\,(\forall g \to d \in R \text{ s.t. } \mathcal{V}ar(\bar{l}) \cap \mathcal{V}ar(\bar{g}) = \emptyset) \\
 & & (\forall \sigma : \mathcal{V}ar(\bar{l}|_p, \bar{g}) \to \mathcal{T}(\mathcal{F}, \mathcal{X}) \text{ s.t. } \bar{l}|_p\sigma = \bar{g}\sigma)\,\text{SOF}(l|_p) \wedge \text{SOF}(g) \\
\text{SOF}(u) & := & (\forall q \in \mathcal{FP}os(u)\backslash\{\Lambda\})\,\text{OF}(u|_q) \\
\text{OF}(v) & := & (\forall g \to d \in R \text{ s.t. } \mathcal{V}ar(\overline{v}) \cap \mathcal{V}ar(\overline{g}) = \emptyset) \\
 & & (\forall o \in \mathcal{FP}os(v))(\forall \sigma : \mathcal{V}ar(\overline{v}, \overline{g}) \to \mathcal{T}(\mathcal{F}, \mathcal{X}))\ \overline{v}|_o\sigma \neq \overline{g}\sigma
\end{array}$$

SOF stands for *subterm overlap-free*, and OF for *overlap-free*. In words, if two lefthand sides of rules in R overlap (linearly) a lefthand side l of a rule in R at positions p and q respectively, then either $p = q$ or $p\#q$. Overlaps at different positions along a path from the root to a leaf of l are forbidden.

Layered systems is a decidable class that relates to overlay systems [6], for which overlaps computed with plain unification can only take place at the root of terms –hence their name–, and generalizes strongly non-overlapping systems [24] which admit no linearized overlaps at all. All these classes are Turing-complete since they contain a complete class [16].

▶ **Example 20.** NKH is a layered system, which is also overlay. $\{h(f(x,y)) \to a, f(x,c(x)) \to b\}$ is layered but not overlay. $\{h(f(x,x)) \to a, f(x,c(x)) \to b, g \to c(g)\}$ is layered, but not strongly non-overlapping. $\{f(h(x)) \to x, h(a) \to a, a \to b\}$ is not overlay nor layered: SOF$(h(x))$ succeeds while SOF$(h(a))$ fails, hence their conjunction fails.

6.1 Layering

We define the rank of a term t as the maximum number of non-overlapping linearized redexes traversed from the root to some leaf of t, which differs from the usual redex-depth.

▶ **Definition 21.** Given a layered rewrite system R, the *rank* $rk(t)$ of a term t is defined by induction on the size of terms as follows:

- the maximal rank of its immediate subterms if t is not a linearized redex ; otherwise,
- 1 plus $max\{rk(\sigma) : (\exists l \to r \in R)\, t = \bar{l}\sigma\}$, where $rk(\sigma) := max\{rk(\sigma(x)) : x \in \mathcal{V}ar(\bar{l})\}$.

▶ **Definition 22.** A rewrite system R is *rank non-increasing* if for all terms u, v such that $u \underset{R}{\to} v$, then $rk(u) \geq rk(v)$.

The rewrite system $\{f(x) \to c(f(x))\}$ is rank non-increasing while $\{f(x) \to f(f(x))\}$ is rank increasing. The system $\{fib(0) \to 0, fib(S(0)) \to S(0), fib(S(S(x))) \to fib(S(x)) + fib(x)\}$ calculating the Fibonacci function is rank non-increasing. NKH is rank non-increasing. The coming decidable sufficient condition for rank non-increasingness captures our examples (but Fibonacci, for which an even more complex decidable property is needed):

▶ **Lemma 23.** *A layered rewrite system R is rank non-increasing if each rule $g \to d$ in R satisfies the following properties:*

(i) $((\forall l \to r \in R)(\forall l' \to r' \in R)$ *s.t.* $\mathcal{V}ar(d), \mathcal{V}ar(\overline{l}), \mathcal{V}ar(\overline{l'})$ *are pairwise disjoint)*
$(\forall p, q \in \mathcal{FP}os(d)$ *s.t.* $q > p \cdot \mathcal{FP}os(l))$
$(\forall \sigma : \mathcal{V}ar(g, \overline{l}, \overline{l'}) \to \mathcal{T}(\mathcal{F}))\ (d|_p\sigma \neq \overline{l}\sigma) \vee (d|_q\sigma \neq \overline{l'}\sigma)$;

(ii) $((\forall l \to r \in R)$ *s.t.* $\mathcal{V}ar(g) \cap \mathcal{V}ar(\overline{l}) = \emptyset)(\forall p \in \mathcal{FP}os(l) \setminus \Lambda)$
$(\forall \sigma : \mathcal{V}ar(g, \overline{l}) \to \mathcal{T}(\mathcal{F})$ *s.t.* $d\sigma = \overline{l}|_p\sigma)$
$((\exists l' \to r' \in R)$ *s.t.* $\mathcal{V}ar(\overline{l'}) \cap \mathcal{V}ar(g, \overline{l}) = \emptyset)(\exists x \notin \mathcal{V}ar(\overline{l}, \overline{l'}, g))\ \overline{l}[x]_p \trianglerighteq \overline{l'}$.

We can now index term-related notions by the rank of terms. Let $\mathcal{T}_n(\mathcal{F}, \mathcal{X})$ (in short, $\mathcal{T}_n$) be the set of terms of rank at most n. Two terms in $\mathcal{T}_n$ are *n-convertible* (resp. *n-joinable*) if their R-conversion (resp. R-joinability) involves terms in $\mathcal{T}_n$ only.

6.2 Closure properties

Call a term u an *OF-term* if u satisfies $\mathrm{OF}(u)$, and a substitution an *OF-substitution* if it maps variables to OF-terms. OF-terms enjoy several important closure properties. Given two substitutions θ, σ and rank n, let

$\mathrm{Conv}_n^\theta(\overline{u},\overline{v})$ iff $\overline{u}\theta$ and $\overline{v}\theta$ are n-convertible, and
$\mathrm{Equalize}_n(\overline{u})_\sigma^\theta$ iff $\overline{u}\theta \twoheadrightarrow_{R_R} u\sigma$ with terms of rank at most n.

▶ **Lemma 24.** *For all OF-terms u and substitutions γ, $u\gamma$ cannot sub-rewrite at a position $p \in \mathcal{FP}os(u)$.*

▶ **Corollary 25.** *OF-terms are preserved under instantiation by OF-substitutions.*

▶ **Lemma 26.** *Let u, v be two terms such that $Conv_n^\theta(\overline{u},\overline{v})$, $Equalize_n(\overline{u})_\sigma^\theta$ and $Equalize_n(\overline{v})_\sigma^\theta$. Then $u\sigma$ and $v\sigma$ are n-convertible.*

▶ **Lemma 27.** *Let $\wedge_i u_i = v_i$ be obtained by decomposition of a unification problem P. Assume all equations $u_i = v_i$ satisfy the properties $Conv_n^\theta(\overline{u_i},\overline{v_i})$, $Equalize_n(\overline{u_i})_\sigma^\theta$, $Equalize_n(\overline{v_i})_\sigma^\theta$, $OF(u_i)$ and $OF(v_i)$. Assume further that n-convertible terms are joinable. Then, unification of P succeeds, and returns a solved form whose all equations satisfy these five properties.*

In this lemma and coming proof, we assume that linearizations are propagated by the unification rules, implying in particular that $\overline{u|_p} = \overline{u}|_p$. P defines the initial linearization.

Proof. We show that these five properties are invariant by the unification rules. The claim follows since the unification rules terminate. We use notations of Figure 1.

- **Remove**, **Choose**, **Swap** are straightforward.
- **Decomp**. By assumption, $\mathrm{Conv}_n^\theta(\overline{f(\vec{s})},\overline{f(\vec{t})})$, hence $\overline{f(\vec{s})}\theta$ and $\overline{f(\vec{t})}\theta$ are joinable by using terms of rank at most n, since R is rank non-increasing. By assumption $\mathrm{OF}(f(\vec{s}))$ and $\mathrm{OF}(f(\vec{t}))$, hence no rewrite can take place at the root. The result follows.
- **Conflict**. By the same token, $f = g$, a contradiction. Thus **Conflict** is impossible.

- **Coalesce**. By assumption, $\mathrm{Conv}_n^\theta(x^k,y^l)$, $\mathrm{Equalize}_n(x^k)_\sigma^\theta$, $\mathrm{Equalize}_n(y^l)_\sigma^\theta$, and $(\forall u{=}v \in P)$, $\mathrm{Conv}_n^\theta(\overline{u},\overline{v})$, $\mathrm{OF}(u)$, $\mathrm{Equalize}_n(\overline{u})_\sigma^\theta$, $\mathrm{OF}(v)$ and $\mathrm{Equalize}_n(\overline{v})_\sigma^\theta$. Putting these things together, we get $\mathrm{Conv}_n^\theta(\overline{u}\{x^k \mapsto y^l\},\overline{v}\{x^k \mapsto y^l\})$, hence $\mathrm{Conv}_n^\theta(\overline{u\{x \mapsto y\}},\overline{v\{x \mapsto y\}})$. Similarly, properties $\mathrm{Equalize}_n(\overline{u\{x \mapsto y\}})_\sigma^\theta$ and $\mathrm{Equalize}_n(\overline{v\{x \mapsto y\}})_\sigma^\theta$ hold. Property $\mathrm{OF}(u)$ is of course preserved by variable renaming for any u.
- **Merge**. Assume $\mathrm{Conv}_n^\theta(x^k,\overline{s})$, $\mathrm{Conv}_n^\theta(x^l,\overline{t})$, $\mathrm{OF}(s)$, $\mathrm{Equalize}_n(\overline{s})_\sigma^\theta$, $\mathrm{Equalize}_n(x^k)_\sigma^\theta$, $\mathrm{OF}(t)$, $\mathrm{Equalize}_n(\overline{t})_\sigma^\theta$ and $\mathrm{Equalize}_n(x^l)_\sigma^\theta$. $\mathrm{Conv}_n^\theta(\overline{s},\overline{t})$ follows from $\mathrm{Conv}_n^\theta(x^k,\overline{s})$, $\mathrm{Conv}_n^\theta(x^l,\overline{t})$, $\mathrm{Equalize}_n(x^k)_\sigma^\theta$ and $\mathrm{Equalize}_n(x^l)_\sigma^\theta$. The other properties follow similarly.
- **Replace**. The proof is similar for the first 3 properties. Further, OF is preserved by replacement by Corollary 25.
- **Merep**. Similar to **Merge**. ◀

▶ **Example 28** (NKH). Let $P = f(x,x) = f(y,c(y))$. Then $P \to_{\mathbf{Decomp}} x = y \wedge x = c(y) \to_{\mathbf{Replace}} c(y) = y \wedge x = c(y) \to_{\mathbf{Swap}} y = c(y) \wedge x = c(y)$. Successive linearizations yield $f(x^1,x^2){=}f(y^1,c(y^2))$, $x^1{=}y^1 \wedge x^2{=}c(y^2)$, $c(y^2){=}y^1 \wedge x^2{=}c(y^2)$ and $y^1{=}c(y^2) \wedge x^2{=}c(y^2)$. The announced properties of the solved form can be easily verified.

▶ **Corollary 29.** *Let $l \to r, g \to d \in R$ and $p \in \mathcal{FP}os(l)$ such that $\mathcal{V}ar(l) \cap \mathcal{V}ar(g) = \emptyset$, and $\overline{l}|_p\theta = \overline{g}\theta$ are terms in $\mathcal{T}_{n+1}$. Then, unification of $l|_p{=}g$ succeeds, returning a solved form S s.t., for each $z = s \in S$, $Conv_n^\theta(\overline{z},\overline{s})$, $OF(s)$, $Equalize_n(\overline{s})_\sigma^\theta$ for all σ satisfying $(\overline{l}\theta \twoheadrightarrow^{(>\mathcal{FP}os(l))} l\sigma) \wedge (\overline{g}\theta \twoheadrightarrow^{(>\mathcal{FP}os(g))} g\sigma)$, and further, $SOF(l|_p\eta_S) \wedge SOF(g\eta_S)$.*

Proof. Unification applies first **Decomp**. Conclude by Lemmas 27 and Corollary 25. ◀

▶ **Corollary 30.** *Assume $t = \overline{l}\sigma$ for some $l \to r \in R$. Then, $rk(t) = 1 + rk(\sigma)$.*

Proof. Let $t = \overline{l_i}\sigma_i = \overline{l_i}\theta\gamma$ (note that γ does not depend on i), where $\theta = mgu(=_i \overline{l_i})$. Then, $rk(t) = 1 + max_i\{rk(\sigma_i)\} = 1 + max_i\{rk(\theta\gamma)\} = 1 + rk(\gamma) = 1 + rk(\sigma_i)$ since θ satisfies OF at all non-variable positions by Lemma 27. ◀

▶ **Example 31** (NKH). Consider $f(c(g),c(g))$ of rank 2, using either linearized lefthand side $f(x^1,x^2)$ or $f(y^1,c(y^2))$ to match $f(c(g),c(g))$. Corresponding substitutions have rank 1.

A major consequence is that the preparatory phase of sub-rewriting operates on terms of a strictly smaller rank. This would not be true anymore, of course, with a conversion-based preparatory phase. More generally, we can also show that the rank of terms does not increase –but may remain stable– when taking a subterm, a property which is not true of non-layered systems. Consider the system $\{f(g(h(x))) \to x, g(x) \to x, h(x) \to x\}$. The redex $f(g(h(a)))$ has rank 1 with our definition, but its subterm $g(h(a))$ has rank 2.

6.3 Testing confluence of layered systems via their cyclic critical pairs

Since R is rank non-increasing we shall prove confluence by induction on the rank of terms. Since rewriting is rank non-increasing, the set of $\mathcal{T}_n$-conversions is closed under diagram rewriting, hence allowing us to use Corollary 5. This is why we adopted this restricted, but complete, form of decreasing diagram rather than the more general form described in [29].

▶ **Definition 32** (Cyclic critical pairs). Given a layered rewrite system R, let $l \to r, g \to d \in R$ and $p \in \mathcal{FP}os(l)$ such that $\mathcal{V}ar(l) \cap \mathcal{V}ar(g) = \emptyset$, and $l|_p = g$ is unifiable with canonical cyclic unifier $\langle \eta_S = \{\vec{x} \mapsto \vec{u}\}, R_S = \{\vec{y} \to \vec{v}\}\rangle$. Then, $r\eta_S \;{}_R\!\!\leftarrow l\eta_S =^{cc}_{R_S\eta_S} l[g]_p\eta_S \to_R l[d]_p\eta_S$ is a *cyclic critical peak*, and $\langle r\eta_S, l[d]_p\eta_S\rangle$ is a *cyclic critical pair*, which is said to be *realizable* by the substitution θ iff $(\forall y \to v \in R_S)\, y\theta \twoheadrightarrow_R {}_R\!\!\twoheadleftarrow v\theta$.

The relationship between critical peaks and realizable cyclic critical pairs, usually called critical pair lemma, is more complex than usual:

▶ **Lemma 33** (Cyclic critical pair lemma). *Let $l \to r, g \to d \in R$ such that $\mathcal{V}ar(l) \cap \mathcal{V}ar(g) = \emptyset$. Let $r\sigma \overset{\Lambda}{_{l\to r}}\!\!\leftarrow l\sigma \overset{(>\mathcal{FP}os(l))}{\twoheadleftarrow} \bar{l}\theta = \bar{l}\theta[\bar{g}\theta]_p \twoheadrightarrow^{(>p\cdot\mathcal{FP}os(g))} \bar{l}\theta[g\sigma]_p \to^p_{g\to d} \bar{l}\theta[d\sigma]_p$ be a sub-rewriting local peak in $\mathcal{T}_{n+1}$, satisfying $p \in \mathcal{FP}os(l)$ and $\mathcal{V}ar(\bar{l}\theta) \cap \mathcal{V}ar(l, g) = \emptyset$. Assume further that R is Church-Rosser on the set $\mathcal{T}_n$. Then, there exists a cyclic solution $\langle \gamma, R_S\rangle$ such that S is a solved form of the unification problem $l|_p = g$, $\gamma = \eta_S\rho$ for some ρ of domain included in $\mathcal{V}ar(l, g)$, $\sigma \twoheadrightarrow_R \gamma$, and R_S is realizable by γ.*

Proof. Corollary 29 asserts the existence of a solved form $S = \vec{x} = \vec{u} \wedge \vec{y} = \vec{v}$ of the problem $l|_p = g$. But $\langle \sigma, R_S\rangle$ may not be a cyclic solution. We shall therefore construct a new substitution γ such that $\sigma \twoheadrightarrow_{R_R} \gamma$ and $\langle \gamma, R_S\rangle$ is a cyclic solution of the problem, obtained as an instance by some substitution ρ of the most general cyclic unifier $\langle \eta_S, R_S\rangle$ by Theorem 18.

The construction of γ has two steps. The first aims at forcing the equality constraints given by S. This step will result in each parameter having possibly many different values. The role of the second step will be to construct a single value for each parameter.

We start equalizing independently equations $z = s \in S$. Since $\mathrm{Equalize}_n(z^j)^\theta_\sigma$, $\mathrm{Equalize}_n(\bar{s})^\theta_\sigma$ and $\mathrm{Conv}^\theta_n(z^j, \bar{s})$, $z\sigma$ and $s\sigma$ are n-convertible by Lemma 26. By assumption, $z\sigma$ and $s\sigma$ are joinable, hence there exists a term t^s_z such that $z\sigma \twoheadrightarrow_R t^s_z \overset{}{_R}\!\!\twoheadleftarrow s\sigma$. Since $\mathrm{OF}(s)$ by Corollary 29, the derivation from $s\sigma$ to t^s_z must occur at positions below $\mathcal{FP}os(s)$. Maintaining equalities in $s\sigma$ between different occurrences of each variable in $\mathcal{V}ar(s)$, we get $t^s_z = s\tau^s_z$ for some τ^s_z. For each parameter p, $p\sigma \twoheadrightarrow_R p\tau^s_z$, hence the elements of the non-empty set $\{p\tau^s_z : p \in \mathcal{V}ar(s) \text{ for some } z = s \in S\}$ are n-convertible thanks to rank non-increasingness. By our Church-Rosser assumption, they can all be rewritten to a same term t_p. We now define γ:

- **(i)** parameters. Given $p \in \mathcal{P}$, we define $\gamma(p) = t_p$. By construction, $p\sigma \twoheadrightarrow_R t_p = p\gamma$.
- **(ii)** finite variables. Given $x = u \in \vec{x} = \vec{u}$, let $\gamma(x) = u\gamma_{|\mathcal{P}}$, thus $x\gamma = u\gamma$. By construction, $x\sigma \twoheadrightarrow_R u\tau^u_x \twoheadrightarrow_R u\gamma$, hence $x\sigma \twoheadrightarrow_R x\gamma$.
- **(iii)** cyclic variables. Given $y = v \in \vec{x} = \vec{u}$, let $\gamma(y) = y\sigma$, making $y\sigma \twoheadrightarrow_R y\gamma$ trivial.
- **(iv)** variables in $\mathcal{V}ar(l, g) \setminus \mathcal{V}ar(l|_p, g)$, that is, those variables from the context $l[\cdot]_p$ which do not belong to the unification problem $l|_p = g$, hence to the solved form S. Given $z \in \mathcal{V}ar(l, g) \setminus \mathcal{V}ar(l|_p, g)$, let $\gamma(z) = z\sigma$, making $z\sigma \twoheadrightarrow_R z\gamma$ trivial.

Therefore $\sigma \twoheadrightarrow_R \gamma$. We proceed to show $\langle \gamma, R_S\rangle$ is a cyclic solution of $l|_p = g$. Take $\rho = \gamma|_{\neg\vec{x}}$. It is routine to see $\gamma = \eta_S\rho$, and to check that $\langle \eta_S, R_S\rangle$ is a cyclic unifier of S by Definition 14, hence of $l|_p = g$ by Lemma 17. Hence the statement.

We end up the proof by noting that γ is a realizer of R_S. ◀

In case of NKH, the lemma is straightforward since solved forms have no parameters.

Our proof strategy for proving confluence of layered systems is as follows: assuming that n-convertible terms are joinable, we show that $(n+1)$-convertible terms are $(n+1)$-joinable by exhibiting appropriate decreasing diagrams for all their local peaks. To this end, we need to define a labelling schema for sub-rewriting. Assuming that rules have an integer index, different rules having possibly the same index, a step $u \overset{p}{\underset{R_R}{\to}} v$ with the rule $l_i \to r_i$ is labelled by the pair $\langle rk(u|_p), i\rangle$. Pairs are compared in the order $\trianglerighteq = (\geq_N, \geq_N)_{lex}$ whose strict part is well-founded. Indexes give more flexibility (shared indexes give even more) in finding decreasing diagrams for critical pairs, this is their sole use.

▶ **Definition 34.** Let $l \to_i r, g \to_j d \in R$ and $p \in \mathcal{FP}os(l)$ such that $l|_p = g$ has a solved form S. Then, the cyclic critical pair $\langle r\eta_S, l[d]_p\eta_S\rangle$ has a *cyclic-joinable decreasing diagram* if

$r\eta_S \overset{\langle 1,I\rangle}{\underset{R}{\twoheadrightarrow}} s =^{cc}_{R_S\eta_S} t \overset{\langle 1,J\rangle}{\underset{R}{\twoheadleftarrow}} l[d]_p\eta_S$, whose sequences of indexes I and J satisfy the decreasing diagram condition, with the additional condition, in case $\mathcal{V}ar(l[\cdot]_p) \neq \emptyset$, that all steps have a rule index $k < i$.

By Corollary 29, the ranks of $l\eta_S$ and $l[g]_p\eta_S$ are 1. Thanks to rank non-increasingness and Definition 21, the cyclic-joinable decreasing diagram –but the congruence closure part– is made of terms of rank 1 except possibly s and t which may have rank 0. It follows that all redexes rewritten in the diagram have rank 1. The decreasing diagram condition is therefore ensured by the rule indexes, which justifies our formulation.

Note further that the condition $\mathcal{V}ar(l[\cdot]_p) = \emptyset$ is automatically satisfied when $p = \Lambda$, hence no additional condition is needed in case of a root overlap. In case where $\mathcal{V}ar(l[\cdot]_p) \neq \emptyset$, implying a non-root overlap, the additional condition aims at ensuring that the decreasing diagram is stable under substitution. It implies in particular that there exists no i-facing step. This may look restrictive, and indeed, we are able to prove a slightly better condition: (i) there exists no i-facing step, and (ii) each step $u \to^q_k v$ using rule k at position q satisfies $k < i$ or $\mathcal{V}ar(u|_q) \subseteq \mathcal{V}ar(g\eta_S)$. We will restrict ourselves here to the simpler condition which yields a less involved confluence proof.

We can now state and prove our main result:

▶ **Theorem 35.** *Rank non-increasing layered systems are confluent provided their realizable cyclic critical pairs have cyclic-joinable decreasing diagrams.*

Proof. Since $\to_R \subseteq \to_{R_R} \subseteq \underset{R}{\twoheadrightarrow}$, R-convertibility and R_R-convertibility coincide. We can therefore apply van Oostrom's theorem to R_R-conversions, and reason by induction on the rank. We proceed by inspection of the sub-rewriting local peaks $v \,{}^{p}_{(l\to r)_R}\!\!\leftarrow u \to^{q}_{(g\to d)_R} w$, with $\mathcal{V}ar(l) \cap \mathcal{V}ar(g) = \emptyset$. We also assume for convenience that $\mathcal{V}ar(l,g) \cap \mathcal{V}ar(u,v,w) = \emptyset$. This allows us to consider u, v, w as ground terms by adding their variables as new constants. We assume further that variables $x, y \in \mathcal{V}ar(l,g)$ become linearized variables x^i, y^j in $\bar{l}, \bar{g}$, and that ξ is the substitution such that $\xi(x^i) = x$ and $\xi(y^j) = y$, hence implying $\mathcal{V}ar(\bar{l}) \cap \mathcal{V}ar(\bar{g}) = \emptyset$.

By definition of sub-rewriting, $u|_p = \bar{l}\theta \underset{R}{\twoheadrightarrow}^{(>\mathcal{FP}os(l))} v'|_p = l\sigma$ and $v = u[r\sigma]_p$, where for all positions $o \in \mathcal{P}os(l)$ such that $l|_o = x$ and $\bar{l}|_o = x^i$, then $x^i\theta \twoheadrightarrow_R x\sigma$. Similarly, $u|_q = \bar{g}\theta \underset{R}{\twoheadrightarrow}^{(>\mathcal{FP}os(g))} w'|_q = g\sigma$ and $w = u[d\sigma]_q$, where for all positions $o \in \mathcal{P}os(g)$ such that $g|_o = y$ and $\bar{g}|_o = y^j$, then $y^j\theta \twoheadrightarrow_R y\sigma$. There are three cases:

1. $p \# q$. The case of disjoint redexes is as usual.
2. $q > p \cdot \mathcal{FP}os(l)$, the so-called ancestor peak case, for which sub-rewriting shows its strength. W.l.o.g. we assume $u|_p$ has some rank $n+1$ and note that $u|_q$ has some rank $m \leq n$ by Corollary 30. Since the sub-rewriting steps from u to w occur strictly below $p \cdot \mathcal{FP}os(l)$, then $q = p \cdot o \cdot q'$ where $l|_o = \xi(y^j)$ and $\bar{l}|_o = y^j$. It follows that $w = \bar{l}\tau$ for some τ which is equal to θ for all variables in $\bar{l}$ except y^j for which $\tau(y^j) = \theta(y^j)[d\sigma]_{q'}$. We proceed as follows: we equalize all n-convertible terms $\{x\sigma : x \in \mathcal{V}ar(r)\}$ in v and $\{y\tau : y \in \mathcal{V}ar(\bar{l})\}$ in w by induction hypothesis, yielding s, t. Note that steps from v to s have ranks strictly less than the rank $n+1$ of the step $u \to_{R_R} v$ by Corollary 30 and rank non-increasingness. Then, t is an instance of l by some γ, and s is the corresponding instance of r, hence t rewrites to s with $l \to r$. The equalization steps from w to t have ranks which are not guaranteed to be strictly less than m, hence cannot be kept to build a decreasing diagram. But they can be absorbed in a sub-rewriting step from w to s whose first label is at most $n+1$, hence faces the step from $u \to_{R_R} v$: sub-rewriting allows us to rewrite directly from w to s, short-cutting the rewrites from w to t that would otherwise

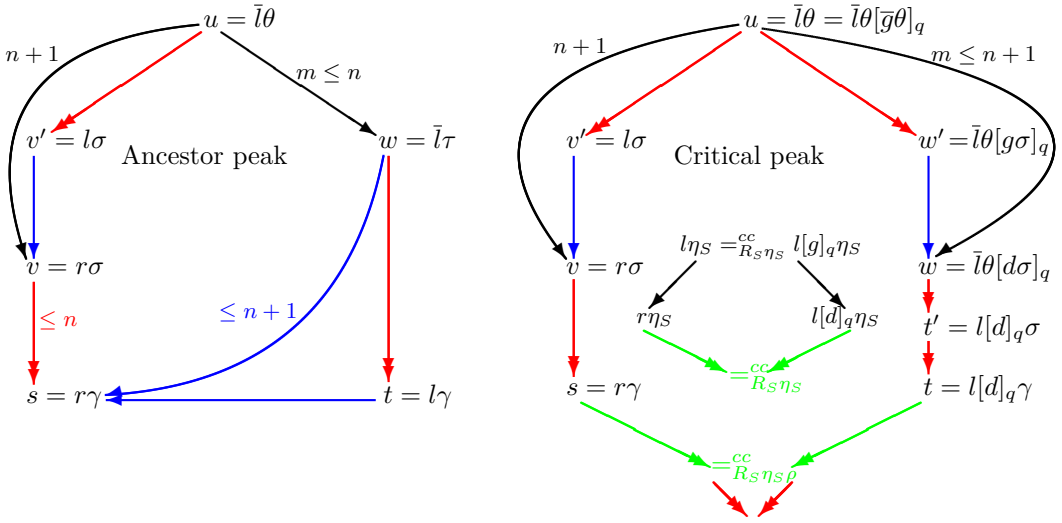


Figure 2 Ancestor and Critical Peaks.

yield a non-decreasing diagram. The proof is depicted at Figure 2 (left), assuming $p = \Lambda$ for simplicity. Black color is used for the given sub-rewriting local peak, blue for arrows whose redexes have ranks at most $n+1$, and red when redex has rank at most n.

3. $q \in p \cdot \mathcal{FP}os(l)$, the so-called critical peak case, whose left and right rewrite steps have labels $\langle n+1, i\rangle$ and $\langle m, j\rangle$ respectively, with rules $l \to r$ and $g \to d$ having indexes i and j. Assuming without loss of generality that $p = \Lambda$, the proof is depicted at Figure 2 (right). Most technical difficulties here originate from the fact that the context $l[\cdot]_q$ may have variables. In this case, we first rewrite w to $t' = l\sigma[d\sigma]_q = l[d]_q\sigma$ by replaying those equalization steps, of rank at most n, used in the derivation from u to v', which apply to variable positions in $\mathcal{V}ar(\bar{l}[\cdot]_q)$.
   Now, since $\bar{l}\theta = \bar{l}\theta[\bar{g}\theta]_q$, by Lemma 33, there is a substitution γ and a solved form S of the unification problem $l|_q = g$, such that $\sigma \twoheadrightarrow_R \gamma$, $\gamma = \eta_S\rho$ for some ρ, and R_S is realizable by γ. By assumption, the cyclic critical pair $\langle r\eta_S, l[d]_q\eta_S\rangle$ has a cyclic-joinable decreasing diagram (modulo $=^{cc}_{R_S\eta_S}$). We can now lift this diagram to the pair $\langle s, t\rangle$ by instantiation with the substitution ρ. The congruence closure used in the lifted diagram becomes therefore $=^{cc}_{R_S\eta_S\rho}$. We are left showing that the obtained diagram for the pair $\langle v, w\rangle$ is decreasing with respect to the local peak $v \leftarrow u \to w$.
   This diagram is made of three distinct parts: the equalization steps, the rewrite steps instantiating the cyclic-joinability assumption with ρ, which originate from s and t –we call them the middle part–, and the congruence closure steps. By Corollary 30, the left equalization steps $v = r\sigma \twoheadrightarrow_R r\gamma = s$ use rewrites with redexes of rank at most n, hence their labels are strictly smaller than $\langle n+1, i\rangle$. The right equalization steps $w \twoheadrightarrow t' \twoheadrightarrow t$ are considered together with the (green-)middle-part rewrite steps. There are two cases depending on whether $l[\cdot]_q$ is variable-free or not:
   (a) $\mathcal{V}ar(l[\cdot]_q) = \emptyset$, hence $m = n+1$ by Corollary 30. In this case, $w = t'$, and by Corollary 30, the rewrite steps $w = l[d]_q\sigma \twoheadrightarrow_R l[d]_q\gamma = t$ have redexes of rank at most n, making their labels strictly smaller than $\langle m, j\rangle = \langle n+1, j\rangle$. Let us now consider the middle-part rewrite steps. Thanks to rank non-increasingness, all terms in this part have rank at most $n+1$. It follows that the associated labels are pairs of the form $\{\langle n', i'\rangle : n' \leq n+1, i' \in I\}$ on the left, or $\{\langle n', j'\rangle : n' \leq n+1, j' \in J\}$ on the right. The assumption that I, J satisfy the decreasing diagram condition for the

critical peak ensures that these rewrites do satisfy the decreasing diagram condition with respect to the local peak $v \leftarrow u \rightarrow w$ as well.

(b) $\mathcal{V}ar(l[\cdot]_q) \neq \emptyset$. By Corollary 30, the right equalization steps $w \twoheadrightarrow t' \twoheadrightarrow t$ have redexes of rank at most n, making their labels strictly smaller than $\langle n+1, i\rangle$. Consider now the middle part. Thanks to rank non-increasingness and the additional condition on the cyclic-joinability assumption of the cyclic critical pair, all labels $\langle n', k\rangle$ in the middle part satisfy $n' \leq n+1$ and $k < i$, hence are strictly smaller than $\langle n+1, i\rangle$.

We are left with the congruence closure steps. Given $y = v \in R_S$, $y\gamma \twoheadrightarrow_R {}_R\!\!\twoheadleftarrow v\gamma$ since R_S is realizable by γ. By Lemma 27, $\mathrm{OF}(v)$ holds, hence $y\gamma$ and $v\gamma$ are n-convertible by rank non-increasingness. We are left with replacing the $=^{cc}_{\vec{y}\gamma=\vec{v}\gamma}$-steps by a joinability diagram whose all steps have rank at most n. The obtained diagram is therefore decreasing, which ends the proof. ◀

Using the improved condition of cyclic-joinability mentioned after Definition 34 requires modifying the discussion concerning the (green-)middle-part rewrite steps. Although this does not cause any conceptual difficulties, it is technically delicate. The interested reader can of course reconstruct this proof for himself or herself.

Our result gives an answer to NKH: confluence of critical pair free rewrite systems can be analyzed via their sub-rewriting critical pairs, which are actually the cyclic critical pairs.

NKH is critical pair free but non-confluent. Indeed, it has the Ω solved form $x = c(y) \wedge y = c(y)$ obtained by unifying $f(x,x) = f(y, c(y))$. The cyclic critical peak is then $a \leftarrow f(x,x) =^{cc} f(y,c(y)) \rightarrow b$ yielding the cyclic critical pair $\langle a, b\rangle$ which is not joinable modulo $\{x = c(y), y = c(y)\}$.

We now give a slight modification of NKH making it confluent:

▶ **Example 36.** The system $R = \{f(x,x) \rightarrow_2 a(x,x),\ f(x,c(x)) \rightarrow_2 b(x),\ f(c(x),c(x)) \rightarrow_3 f(x,c(x)),\ a(x,x) \rightarrow_1 e(x),\ b(x) \rightarrow_1 e(c(x)),\ g \rightarrow_0 c(g)\}$ is confluent. Showing that R satisfies (DLO) is routine, and it is rank non-increasing by Lemma 23. There are three cyclic critical pairs, which all have a cyclic-joinable decreasing diagram. For instance, the unification $f(x,x) = f(y,c(y))$ returns a canonical cyclic unifier $\langle \eta_S = \emptyset, R_S = \{x \rightarrow c(y), y \rightarrow c(y)\}\rangle$, the corresponding cyclic critical peak $a(x,x) \ {}^{\langle 1,2\rangle}\!\leftarrow f(x,x) =^{cc}_{R_S\eta_S} f(y,c(y)) \rightarrow^{\langle 1,2\rangle} b(y)$ has a cyclic-joinable decreasing diagram $a(x,x) \rightarrow^{\langle 1,1\rangle} e(x) =^{cc}_{R_S\eta_S} e(c(y)) \ {}^{\langle 1,1\rangle}\!\leftarrow b(y)$. The unification $f(x,x) = f(c(y),c(y))$ returns $\langle \eta_S = \{x = c(y)\}, R_S = \emptyset\rangle$, the corresponding (normal) critical peak $a(c(y),c(y)) \ {}^{\langle 1,2\rangle}\!\leftarrow f(c(y),c(y)) \rightarrow^{\langle 1,3\rangle} f(y,c(y))$ decreases by $a(c(y),c(y)) \rightarrow^{\langle 1,1\rangle} e(c(y)) \ {}^{\langle 1,1\rangle}\!\leftarrow b(y) \ {}^{\langle 1,2\rangle}\!\leftarrow f(y,c(y))$. By Theorem 35, R is confluent.

Theorem 35 can be easily used positively: if all cyclic critical pairs have cyclic-joinable decreasing diagrams, then confluence is met. This was the case in Example 36. But there is another positive use that we illustrate now: showing that $\{f(x,x) \rightarrow a, f(x,c(x)) \rightarrow b, g \rightarrow d(g)\}$ is confluent requires proving that the cyclic critical pair given by unifying the first two rules is not realizable. Although realizability is undecidable in general, this is the case here since there is no term s convertible to $c(s)$. Theorem 35 can also be used negatively by exhibiting some realizable cyclic critical pair which is not joinable: this is the case of example NKH. In general, if some realizable cyclic critical pair leading to a local peak is not joinable, then the system is non-confluent. Whether a realizable cyclic critical pair always yields a local peak is still an open problem which we had no time to investigate yet.

A main assumption of our result is that rules may not increase the rank. One can of course challenge this assumption, which could be due to the proof method itself. The following counter-example shows that it is not the case.

▶ **Example 37.** Consider the critical pair free system $R = \{d(x,x) \to 0,\ f(x) \to d(x, f(x)),\ c \to f(c)\}$, which is layered but whose second rule is rank increasing since $d(x^1, x^2)$ unifies with $d(y, f(y))$. This system is non-confluent, since $f(fc) \to d(fc, ffc) \to d(ffc, ffc) \to 0$ while $f(fc) \to f(d(c, fc)) \to f(d(fc, fc)) \to f0$ which generates the regular tree language $\{S \to d(0, S), S \to f0\}$ not containing 0. Note that replacing the second rule by the right linear rule $f(x) \to d(x, f(c))$ yields a confluent system [24].

Releasing rank non-increasingness would indeed require strengthening another assumption, possibly imposing left- or right-linearity.

7 Conclusion

Decreasing diagrams opened the way for generalizing Knuth and Bendix's critical-pair test for confluence to non-terminating systems, re-igniting these questions. Our results answer open problems by allowing non-terminating rules which can also be non-linear on the left as well as on the right. The notion of layered systems is our first conceptual contribution here.

Another, technical contribution of our work is the notion of sub-rewriting, which can indeed be compared to parallel rewriting. Both relations contain plain rewriting, and are included in its transitive closure. Both can therefore be used for studying confluence of plain rewriting. Tait and Martin-Löf's parallel rewriting –as presented by Barendregt in his famous book on Lambda Calculus [2]– has been recognized as the major tool for studying confluence of left-linear non-terminating rewrite relations when they are not right-linear. We believe that sub-rewriting will be equally successful for studying confluence of non-terminating rewrite relations that are not left-linear. In the present work where no linearity assumption is made, assumption (DLO) ensuring the absence of stacked critical pairs in lefthand sides makes the combined use of sub-rewriting and parallel rewriting superfluous. Without that assumption, as is the case in [17], their combined use becomes necessary.

A last contribution, both technical and conceptual, is the notion of cyclic unifiers. Although their study is still preliminary, we have shown that they constitute a powerful new tool to handle unification problems with cyclic equations in the same way we deal with unification problems without cyclic equations, thanks to the existence of most general cyclic unifiers which generalize the usual notion of mgu. This indeed opens the way to a *uniform treatment* of problems where unification, whether finite or infinite, plays a central role.

Our long-term goal goes beyond improving the current toolkit for carrying out confluence proofs for non-terminating rewrite systems. We aim at designing new tools for showing confluence of complex type theories (with dependent types, universes and dependent elimination rules) directly on raw terms, which would ease the construction of strongly normalizing models for typed terms. Since redex-depth, the notion of rank used here, does not behave well for higher-order rules, appropriate new notions of rank are required in that setting.

Acknowledgments. This research is supported in part by NSFC Programs (No. 91218302, 61272002) and MIIT IT funds (Research and application of TCN key technologies) of China, and in part by JSPS, KAKENHI Grant-in-Aid for Exploratory Research 25540003 of Japan.

References

1 Takahito Aoto, Yoshihito Toyama, and Kazumasa Uchida. Proving confluence of term rewriting systems via persistency and decreasing diagrams. In Dowek [7], pages 46–60.

2 Hendrik P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1984.

3 Alain Colmerauer. Equations and inequations on finite and infinite trees. In *FGCS*, pages 85–99, 1984.

4 Hubert Comon, Max Dauchet, Remi Gilleron, Christof Löding, Florent Jacquemard, Denis Lugiez, Sophie Tison, and Marc Tommasi. Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata`, 2007. release October, 12th 2007.

5 Nachum Dershowitz and Jean-Pierre Jouannaud. Rewrite systems. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics (B)*, pages 243–320. North-Holland, 1990.

6 Nachum Dershowitz, Mitsuhiro Okada, and G. Sivakumar. Confluence of conditional rewrite systems. In Stéphane Kaplan and Jean-Pierre Jouannaud, editors, *Conditional Term Rewriting Systems, 1st International Workshop, Orsay, France, July 8-10, 1987, Proceedings*, volume 308 of *Lecture Notes in Computer Science*, pages 31–44. Springer, 1987.

7 Gilles Dowek, editor. *Rewriting and Typed Lambda Calculi – Joint International Conference, RTA-TLCA 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, volume 8560 of *Lecture Notes in Computer Science*. Springer, 2014.

8 Bertram Felgenhauer. Rule labeling for confluence of left-linear term rewrite systems. In *IWC*, pages 23–27, 2013.

9 Jean H. Gallier, Stan Raatz, and Wayne Snyder. Theorem proving using rigid E-unification equational matings. In *Proceedings of the Symposium on Logic in Computer Science (LICS'87), Ithaca, New York, USA, June 22-25, 1987*, pages 338–346. IEEE Computer Society, 1987.

10 Hiroshi Gomi, Michio Oyamaguchi, and Yoshikatsu Ohta. On the Church-Rosser property of root-E-overlapping and strongly depth-preserving term rewriting systems. *IPSJ*, 39(4):992–1005, 1998.

11 Nao Hirokawa and Aart Middeldorp. Decreasing diagrams and relative termination. *J. Autom. Reasoning*, 47(4):481–501, 2011.

12 Gérard P. Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *J. ACM*, 27(4):797–821, 1980.

13 Jean-Pierre Jouannaud and Claude Kirchner. Solving equations in abstract algebras: A rule-based survey of unification. In *Computational Logic – Essays in Honor of Alan Robinson*, pages 257–321, 1991.

14 Jean-Pierre Jouannaud and Vincent van Oostrom. Diagrammatic confluence and completion. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikoletseas, and Wolfgang Thomas, editors, *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part II*, volume 5556 of *Lecture Notes in Computer Science*, pages 212–222. Springer, 2009.

15 Dominik Klein and Nao Hirokawa. Confluence of non-left-linear TRSs via relative termination. In Nikolaj Bjørner and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning – 18th International Conference, LPAR-18, Mérida, Venezuela, March 11-15, 2012. Proceedings*, volume 7180 of *Lecture Notes in Computer Science*, pages 258–273. Springer, 2012.

16 Jan Willem Klop. Term rewriting systems. In *Handbook of Logic in Computer Science, Vol.2*, pages 1–116. Oxford University Press, 1993.

17 Jiaxiang Liu, Nachum Dershowitz, and Jean-Pierre Jouannaud. Confluence by critical pair analysis. In Dowek [7], pages 287–302.

18 Jiaxiang Liu and Jean-Pierre Jouannaud. Confluence: The unifying, expressive power of locality. In Shusaku Iida, José Meseguer, and Kazuhiro Ogata, editors, *Specification,*

Algebra, and Software – Essays Dedicated to Kokichi Futatsugi, volume 8373 of *Lecture Notes in Computer Science*, pages 337–358. Springer, 2014.

19 Ken Mano and Mizuhito Ogawa. Unique normal form property of compatible term rewriting systems: a new proof of Chew's theorem. *Theor. Comput. Sci.*, 258(1-2):169–208, 2001.

20 Kunihiro Matsuura, Michio Oyamaguchi, Yoshikatsu Ohta, and Mizuhito Ogawa. On the E-overlapping property of nonlinear term rewriting systems (in Japanese). *IEICE*, 80-D-I(11):847–855, 1997.

21 Greg Nelson and Derek C. Oppen. Fast decision procedures based on congruence closure. *J. ACM*, 27(2):356–364, 1980.

22 Satoshi Okui. Simultaneous critical pairs and Church-Rosser property. In Tobias Nipkow, editor, *Rewriting Techniques and Applications, 9th International Conference, RTA-98, Tsukuba, Japan, March 30 – April 1, 1998, Proceedings*, volume 1379 of *Lecture Notes in Computer Science*, pages 2–16. Springer, 1998.

23 Barry K. Rosen. Tree-manipulating systems and Church-Rosser theorems. *J. ACM*, 20(1):160–187, 1973.

24 Masahiko Sakai and Mizuhito Ogawa. Weakly-non-overlapping non-collapsing shallow term rewriting systems are confluent. *Inf. Process. Lett.*, 110(18-19):810–814, 2010.

25 Masahiko Sakai, Michio Oyamaguchi, and Mizuhito Ogawa. Non-E-overlapping, weakly shallow, and non-collapsing TRSs are confluent. In *CADE*, 2015. to appear.

26 Terese. Term rewriting systems. In *Cambridge Tracts in Theoretical Computer Science*, volume 55. Cambridge University Press, 2003.

27 Yoshihito Toyama and Michio Oyamaguchi. Church-Rosser property and unique normal form property of non-duplicating term rewriting systems. In Nachum Dershowitz and Naomi Lindenstrauss, editors, *Conditional and Typed Rewriting Systems, 4th International Workshop, CTRS-94, Jerusalem, Israel, July 13-15, 1994, Proceedings*, volume 968 of *Lecture Notes in Computer Science*, pages 316–331. Springer, 1994.

28 Vincent van Oostrom. Confluence by decreasing diagrams. *Theor. Comput. Sci.*, 126(2):259–280, 1994.

29 Vincent van Oostrom. Confluence by decreasing diagrams converted. In Andrei Voronkov, editor, *Rewriting Techniques and Applications, 19th International Conference, RTA 2008, Hagenberg, Austria, July 15-17, 2008, Proceedings*, volume 5117 of *Lecture Notes in Computer Science*, pages 306–320. Springer, 2008.

30 Harald Zankl, Bertram Felgenhauer, and Aart Middeldorp. Labelings for decreasing diagrams. In Manfred Schmidt-Schauß, editor, *Proceedings of the 22nd International Conference on Rewriting Techniques and Applications, RTA 2011, May 30 – June 1, 2011, Novi Sad, Serbia*, volume 10 of *LIPIcs*, pages 377–392. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2011.

A Unified Approach to Boundedness Properties in MSO*

Łukasz Kaiser†[1], Martin Lang[2], Simon Leßenich[3], and Christof Löding[2]

1 **LIAFA, CNRS & Université Paris Diderot – Paris 7, France**
2 **Lehrstuhl für Informatik 7, RWTH Aachen University, Germany**
3 **Mathematische Grundlagen der Informatik, RWTH Aachen University, Germany**

Abstract

In the past years, extensions of monadic second-order logic (MSO) that can specify boundedness properties by the use of operators referring to the sizes of sets have been considered. In particular, the logics costMSO introduced by T. Colcombet and MSO+U by M. Bojańczyk were analyzed and connections to automaton models have been established to obtain decision procedures for these logics. In this work, we propose the logic *quantitative counting MSO* (qcMSO for short), which combines aspects from both costMSO and MSO+U. We show that both logics can be embedded into qcMSO in a natural way. Moreover, we provide decidability proofs for the theory of its weak variant (quantification only over finite sets) for the natural numbers with order and the infinite binary tree. These decidability results are obtained using a regular cost function extension of automatic structures called resource-automatic structures.

1998 ACM Subject Classification F.1.1 Models of Computation, F.4.1 Mathematical Logic, F.4.3 Formal Languages

Keywords and phrases quantitative logics, monadic second order logic, boundedness, automatic structures, tree automata

Digital Object Identifier 10.4230/LIPIcs.CSL.2015.441

1 Introduction

The tight connection of monadic second-order logic (MSO), which is the extension of first-order logic by quantification over sets of elements, and finite automata over word and tree structures has led to a rich theory and many applications and decision procedures in verification and synthesis (see [20] for an introduction and overview).

In the past years, quantitative variants or extensions of MSO with a method to refer to the size of set variables have been introduced. Most prominently, there are costMSO, proposed by T. Colcombet (cf. [11]), and MSO+U, by M. Bojańczyk (cf. [3]). The logic costMSO extends standard MSO by a new atomic formula of the form $|X| \leq N$ (for a set variable X, and a parameter N that is interpreted as natural number) that is only allowed to appear positively in formulas. The N is a global parameter shared among all occurrences of this operator. The logic has a quantitative semantics: In a structure $\mathfrak{S}$, we assign a formula φ the minimal value for $N \in \mathbb{N} \cup \{\infty\}$ such that the formula is satisfied as a normal MSO

* This work was partially supported by the DFG Research Training Group 1298 (AlgoSyn) and the DFG research grant *Automatentheoretische Verifikationsprobleme mit Ressourcenschranken.*
† Currently at Google Inc.

24th EACSL Annual Conference on Computer Science Logic (CSL 2015).
Editor: Stephan Kreutzer; pp. 441–456

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

formula with the additional cardinality bounds. If it cannot be satisfied for any N, the value is ∞ (and if it is satisfiable but no subformula of the form $|X| \leq N$ appears, the value is 0). The logic has proved to be useful for studying boundedness questions, in which the precise values of a formula are not of interest, but rather the question whether the values are bounded on sets of structures (for example on sets of words or trees). This intended semantics of (un)boundedness is directly encoded in MSO+U. It extends standard MSO by a new quantifier $UX.\varphi(X)$. Such a formula is satisfied if there are arbitrarily large finite sets X that satisfy $\varphi(X)$. Correspondingly, the logic MSO+U has a boolean semantics and does not forbid the use of negation.

The algorithmic properties of both logics and equivalent automaton models have been studied intensively. The logic costMSO is closely connected to *regular cost functions* (cf. [10]) and has an automaton model called B-automata. This automaton equivalence proved useful to obtain algorithmic methods for decision procedures of costMSO on finite words and infinite words (cf. [17]) and was also extended to the *weak* variant costWMSO, which only allows set quantification ranging over finite sets, on the infinite binary tree (cf. [21]). Similar questions were investigated for MSO+U. It turned out that the weak variant WMSO+U has equivalent automaton models on infinite words (lookahead limsup automata) (cf. [3]) and on the infinite binary tree (nested limsup automata) (cf. [7]). These yield decision procedures for the theory of WMSO+U on the respective structures. However, very recently M. Bojańczyk, P. Parys and S. Toruńczyk were able to show that the MSO+U-theory of the natural numbers with order is undecidable (cf. [6]).

We aim at constructing an MSO variant that combines the two aspects of costMSO and MSO+U and identified two key ingredients: First, a mechanism to measure the size of sets that satisfy a definable property as provided by $|X| \leq N$ in costMSO. Secondly, the possibility to test for these sizes within the logic similar to the quantifier U in MSO+U. It is clear that one loses decidability very soon if these mechanisms are too precise. Thus, we want to concentrate on the problem of boundedness. We propose the logic *quantitative counting MSO* (for short qcMSO) as a logic with quantitative semantics over the domain $\mathbb{N} \cup \{\infty\}$. Its basic syntax is similar to standard MSO without negation. We add the operator $|X|$ for set variables and the operator $\varphi = \infty$ for formulas. The definition of the semantics is inspired by the quantitative μ-calculus (cf. [14]). We associate `true` with ∞ and `false` with 0. Accordingly, the boolean connectives $\wedge$, $\vee$ are evaluated by min and max. The quantifiers $\exists$, $\forall$ are are evaluated by sup and inf. The formula $|X|$ evaluates to the number of elements of X and $\varphi = \infty$ has the value `true` (∞) if φ evaluates to ∞ and `false` (0) otherwise.

We show that there is a natural translation of costMSO and MSO+U into equivalent qcMSO formulas. Moreover, we show that the questions of *boundedness* and partly also *dominance* that are considered in connection with costMSO formulas can be expressed in qcMSO. These connections also hold for the respective weak variants of the logics.

The main contribution of this work is a decision procedure for qcWMSO sentences over the naturals with linear order and the infinite binary tree. More precisely, we prove the following main theorem.

▶ **Theorem 1.**

(a) *Given a sentence $\varphi \in$ qcWMSO, the evaluation $[\![\varphi]\!]^{(\omega,<)}$ of φ on the natural numbers with order is effectively computable.*

(b) *Given a sentence $\varphi \in$ qcWMSO, it is decidable whether $[\![\varphi]\!]^{\mathfrak{T}_2} = \infty$.*

The theorem is based on a quantitative extension of *automatic structures* called *resource-automatic structures*, which was presented in [18]. This framework introduces structures with quantitative relations – called *resource structures* – and provides a general method to

compute the cost of first-order queries on structures whose relations are representable by B-automata. We show how such first-order queries can be extended with an ∞-comparison operator such that we can translate qcWMSO into such queries. We use the fact that the finite powerset structure of the naturals with a set size relation is resource-automatic. Furthermore, we provide an extension of the framework to finite trees and show that the finite powerset structure of the infinite binary tree with (an approximation of) the set size relation is *resource-tree-automatic*. The obtained decidability result can be seen as the best we could have hoped for as full qcMSO inherits the undecidability of full MSO+U already on the naturals. However, the connection between *weak* qcMSO and resource-automatic structures nicely resembles the known correspondence between WMSO and standard automatic structures.

The remainder of this work is structured as follows: First, we fix some notations and introduce the formal basics of regular cost functions, resource-automatic structures and the logics costMSO and MSO+U. In Section 3, we introduce qcMSO and explain the embedding of costMSO and MSO+U. Section 4 extends the theory of resource-automatic structures such that qcWMSO formulas over the naturals can be expressed in this framework. Moreover, we provide an extension to finite trees that enables us to decide boundedness of qcWMSO even on the infinite binary tree.

2 Preliminaries

We write Σ for a finite alphabet and Σ^* for the set of finite sequences (words) of letters from Σ. To evaluate MSO and its extensions on words, we consider words as relational structures: a word $w = w_1 \dots w_n \in \Sigma^*$ corresponds to the structure $(\{1, \dots, n\}, <, (P_a)_{a\in\Sigma})$, where the P_a are monadic predicates such that $i \in P_a$ if and only if $w_i = a$. For infinite words, that is, sequences from Σ^ω, we use the set ω of natural numbers as the universe. Additionally, we consider finite and infinite binary trees over the alphabet Σ. A tree t is a mapping $\mathrm{dom}(t) \to \Sigma$ where $\mathrm{dom}(t) \subseteq \{0,1\}^*$ is a prefix-closed set that describes the nodes of the tree in such a way that ε represents the root node, and for a node $u \in \{0,1\}^*$, $u0$ is the left and $u1$ the right successor. Since we only consider binary trees, for every $u \in \mathrm{dom}(t)$ we either have $u0, u1 \in \mathrm{dom}(t)$ or $u0, u1 \notin \mathrm{dom}(t)$. A tree is finite if $\mathrm{dom}(t)$ is finite. The set of all finite trees over the alphabet Σ is denoted by $\mathcal{T}_\Sigma$. When we talk about infinite binary trees, we usually mean trees with $\mathrm{dom}(t) = \{0,1\}^*$.

2.1 Regular Cost Functions and the logic costMSO

In [9], regular cost functions were introduced based on two dual variants of *cost automata*. A cost automaton is a normal NFA with an additional finite set Γ of counters. These counters support three kinds of atomic operations: First, a counter can be incremented by one (`i`). Secondly, a counter can be reset to zero (`r`) and lastly, a counter can be *checked* (`c`). The counters are driven by the transitions. Correspondingly, for a cost automaton $(Q, \Sigma, q_0, \Delta, F, \Gamma)$ the transition relation Δ is a subset of $Q \times \Sigma \times Q \times (\{\mathtt{i}, \mathtt{r}, \mathtt{c}\}^*)^\Gamma$. A run ρ of such an automaton is identified with a sequence of states and transitions and is, as usual, called accepting if it starts in q_0 and ends in a state of F. Along the run, we simulate the values of the counters (starting with 0) according to the operations on the transitions, and whenever a counter is checked, its current value is stored for later evaluation. We denote the set of checked counter values (over all counters) by $C(\rho)$. The semantics of cost automata are functions $\Sigma^* \to \mathbb{N} \cup \{\infty\}$, and come in two (dual) flavors: A *B-automaton* has only counter operations of the forms $\{\varepsilon, \mathtt{ic}, \mathtt{r}\}$, a run ρ has the value $\sup C(\rho)$ and a word $w \in \Sigma^*$ is assigned the infimum over all accepting runs ρ on this word. Dually, an *S-automaton* has

only counter operations of the forms $\{\varepsilon, \mathtt{i}, \mathtt{cr}, \mathtt{r}\}$, a run ρ has the value $\inf C(\rho)$ and a word $w \in \Sigma^*$ is assigned the supremum over all accepting runs ρ on this word. For a B-/S-cost automaton $\mathcal{A}$, we write $[\![\mathcal{A}]\!]_B$ and $[\![\mathcal{A}]\!]_S$ to refer to the respective semantics.

Regular cost functions are the functions definable by B- or S-automata up to a certain equivalence relation $\approx$. The equivalence is based on the notion of *correction functions*. A correction function $\alpha : \mathbb{N} \cup \{\infty\} \to \mathbb{N} \cup \{\infty\}$ is a monotone mapping that maps ∞ and only ∞ to ∞. Let $f, g : A \to \mathbb{N} \cup \{\infty\}$ be two functions. We say f is α-dominated by g and write $f \preceq_\alpha g$ if for all $a \in A$: $f(a) \leq \alpha(g(a))$. We call f and g α-equivalent and write $f \approx_\alpha g$ if $f \preceq_\alpha g$ and $g \preceq_\alpha f$. Additionally, we just write $\approx$ to indicate that there exists an α such that the relation holds. The $\approx$ relation is the same as saying that two cost functions are bounded on the same subsets of their domain. Formally, we have $f \approx g$ iff for all $B \subseteq A$: $\sup_{x \in B} f(x) < \infty \Leftrightarrow \sup_{x \in B} g(x) < \infty$. A proof can be found in [9].

Regular cost functions possess closure properties comparable to those of regular languages (cf. [10]). They are closed under min and max of two functions, which extends the classical union and intersection closure. Moreover, they are closed under inf- and sup-projections. These projections extend classical alphabet projection in the following way: Let Σ be an alphabet, $\pi : \Sigma^{k+1} \to \Sigma^k$ a projection that removes the last component and let π^* be the letterwise extension of π to words. The (π)-inf-projection of a function $f : (\Sigma^{k+1})^* \to \mathbb{N} \cup \{\infty\}$ is defined by $w \mapsto \inf_{u \in \pi^{*-1}(w)} f(u)$, and respectively for sup.

It is well-known that regular languages correspond to languages definable in MSO over word models. This equivalence was lifted to regular cost functions with the logic costMSO. The syntax of costMSO is standard MSO syntax extended with a new predicate that is only allowed to appear positively in the formula: $|X| \leq N$ for all set variables X. costMSO is evaluated over standard relational structures by an inductively defined semantics. For the sake of a uniform presentation, we assume (w.l.o.g.) that only set variables are used. Additionally, set inclusion $\subseteq$ is added as a relation. For a relational structure $\mathfrak{S} = (S, R_1, \ldots, R_n)$ and a valuation $\beta : X \to 2^S$ of the free variables the semantics can be defined as follows (see [11]):

$$[\![R_i X_1 \ldots X_{k_i}]\!]^{\mathfrak{S},\beta} := \begin{cases} 0, & (a_1, \ldots, a_{k_i}) \in R_i^{\mathfrak{S}}, \beta(X_i) = \{a_i\} \\ \infty, & \text{otherwise} \end{cases}$$

$$[\![\neg R_i X_1 \ldots X_{k_i}]\!]^{\mathfrak{S},\beta} := \begin{cases} \infty, & (a_1, \ldots, a_{k_i}) \in R_i^{\mathfrak{S}}, \beta(X_i) = \{a_i\} \\ 0, & \text{otherwise} \end{cases}$$

$$[\![|X| \leq N]\!]^{\mathfrak{S},\beta} := |\beta(X)|$$

$$[\![\varphi \wedge \psi]\!]^{\mathfrak{S},\beta} := \max([\![\varphi]\!]^{\mathfrak{S},\beta}, [\![\psi]\!]^{\mathfrak{S},\beta}) \qquad [\![\varphi \vee \psi]\!]^{\mathfrak{S},\beta} := \min([\![\varphi]\!]^{\mathfrak{S},\beta}, [\![\psi]\!]^{\mathfrak{S},\beta})$$

$$[\![\exists X \varphi(X)]\!]^{\mathfrak{S},\beta} := \inf_{S' \subseteq S} [\![\varphi(X)]\!]^{\mathfrak{S},\beta[X \mapsto S']} \qquad [\![\forall X \varphi(X)]\!]^{\mathfrak{S},\beta} := \sup_{S' \subseteq S} [\![\varphi(X)]\!]^{\mathfrak{S},\beta[X \mapsto S']}$$

This semantics assigns each sentence φ a function $\Sigma^* \to \mathbb{N} \cup \{\infty\}$ over word models. The main equivalence theorem for costMSO states that these functions are exactly the regular cost functions (cf. [9]). The central decision problems for costMSO are boundedness and dominance: A formula φ is bounded over a domain $\mathcal{D}$, if there exists a bound $B \in \mathbb{N}$ such that $[\![\varphi]\!]^{\mathfrak{S}} < B$ for all $\mathfrak{S} \in \mathcal{D}$. A formula φ dominates a formula ψ on a domain $\mathcal{D}$, if for all subsets $\mathcal{C} \subseteq \mathcal{D}$ it holds that whenever φ is bounded, ψ is bounded as well.

2.2 MSO+U

Another approach to introduce a method to express boundedness or unboundedness problems in MSO is with the help of the unbounding quantifier. Unlike costMSO, this leads to

a qualitative extension. In MSO+U, a new, third set quantifier U is added to MSO. This quantifier evaluates to true if there are arbitrarily large finite sets that satisfy a formula, so formally, $\mathfrak{A} \vDash_{\text{MSO+U}} UX.\varphi(X)$ if and only if $\mathfrak{A} \vDash_{\text{MSO}} \exists X(|X| > n \wedge |X| < \infty \wedge \varphi(X))$ for every $n \in \mathbb{N}$.

Note that, for every fixed n, $|X| > n$ is expressible in classical MSO. For completeness reasons, the dual quantifier B is also added, with the semantics that $\mathfrak{A} \vDash BX.\varphi$ if and only if there is a (finite) bound on the size of sets that satisfy φ. For obvious reasons, MSO+U is only studied over infinite structures, in particular infinite words and trees.

Unlike classical MSO and costMSO, there is most likely no automaton model for MSO+U with full second-order quantification for topological reasons [15], and furthermore, the MSO+U-theory of the natural numbers with order is already undecidable [6]. However, the weak variant is decidable over infinite words [4] and infinite trees [8].

2.3 (Resource-) Automatic Structures and FO+RR

The theory of automatic structures provides a formalism to obtain logical structures with a decidable first-order theory by automata representations of the structures (for a comprehensive introduction see, e.g., [16]). A relational structure $\mathfrak{S} = (S, R_1, \dots, R_n)$ is called *automatic* if there are a representation of the universe S in form of a regular language and representations of the relations R_1 up to R_n in the form of *synchronous transducers*. A synchronous transducer for a j-ary relation over $S \subseteq \Sigma^*$ can be seen as a (normal) automaton operating over the vector alphabet $(\Sigma \uplus \{\$\})^j$. It reads all j words letter-by-letter in parallel. Shorter words are padded to the length of the longest word with a newly introduced padding symbol \$. This transformation from a tuple of words to a (padded) word over a vector alphabet is called *convolution*.

The main result for automatic structures is that they always have a decidable first-order theory (see [16]). This is obtained by inductively translating logical operations into operations for automata. Boolean connectives can be represented by union and intersection of regular languages and existential quantification corresponds to alphabet projection for vector alphabets. The original result has been extended to ω-words and finite and infinite trees (see [1, 2]).

Motivated by quantitative verification questions, the idea of automatic structures has been lifted to *resource structures* in [18]. Resource structures are a quantitative extension of relational structures. The quantitative aspect is introduced via the relations: A tuple of elements $\bar{a}$ is not just in some relation R or not, but being in relation may *cost* some value from $\mathbb{N} \cup \{\infty\}$. The verification-driven question was how expensive it is to satisfy a first-order definable property in such a structure. The logic FO+RR (first-order over resource relations) was designed to provide a formalism for this question. Its syntax is identical to normal FO without negation. For a resource structure $\mathfrak{S} = (S, R_1, \dots, R_n)$ and a variable interpretation $\beta : X \to S$, the semantics is inductively defined as follows:

$$[\![R_i x_1 \dots x_{k_i}]\!]^{\mathfrak{S},\beta} := R_i^{\mathfrak{S}}(\beta(x_1), \dots, \beta(x_{k_i}))$$

$$[\![x = y]\!]^{\mathfrak{S},\beta} := \begin{cases} 0, & \beta(x) = \beta(y) \\ \infty, & \text{otherwise} \end{cases} \qquad [\![x \neq y]\!]^{\mathfrak{S},\beta} := \begin{cases} \infty, & \beta(x) = \beta(y) \\ 0, & \text{otherwise} \end{cases}$$

$$[\![\varphi \wedge \psi]\!]^{\mathfrak{S},\beta} := \max([\![\varphi]\!]^{\mathfrak{S},\beta}, [\![\psi]\!]^{\mathfrak{S},\beta}) \qquad [\![\varphi \vee \psi]\!]^{\mathfrak{S},\beta} := \min([\![\varphi]\!]^{\mathfrak{S},\beta}, [\![\psi]\!]^{\mathfrak{S},\beta})$$

$$[\![\exists x \varphi(x)]\!]^{\mathfrak{S},\beta} := \inf_{s \in S} [\![\varphi(x)]\!]^{\mathfrak{S},\beta[x \mapsto s]} \qquad [\![\forall x \varphi(x)]\!]^{\mathfrak{S},\beta} := \sup_{s \in S} [\![\varphi(x)]\!]^{\mathfrak{S},\beta[x \mapsto s]}$$

This semantics answers the intuitive question in the following way: Let φ be an FO+RR-formula with $[\![\varphi]\!]^{\mathfrak{S}} = n < \infty$. If we consider φ as a normal FO-formula, it is satisfied if we allow all those tuples to be in relation (in the classical sense) that cost at most n. This relation also explains the absence of negation. In order to preserve this intuitive semantics, monotonicity is necessary: if we allow a higher resource usage, more formulas should become true.

Resource-automatic structures extend the idea of automatic structures to resource structures and FO+RR. A structure is called resource-automatic if its universe can be represented by a regular language and the semantics of the relations can be specified via *synchronous cost transducers*. For the sake of simplicity it suffices to see such a transducer as a B-automaton operating on a vector alphabet in the same way as for standard synchronous transducers. The details can be found in [18]. The main result that we use here is that the value of FO+RR formulas can be computed over resource-automatic structures.

2.4 Cost Tree Automata and Cost Games

In [12], the theory of regular cost functions was lifted to finite trees. Regular cost functions on trees are defined by B- or S-tree automata. These automata can be seen as an extension of B-/S-automata on words and nondeterministic top-down tree automata (see, e.g., [13] for an introduction to regular tree languages and tree automata). For the sake of a simpler presentation, we move the counter actions to the states of the automaton and tailor the definition to our setting of binary trees.

A cost tree automaton is a tuple $\mathcal{A} = (Q, \Sigma, Q_I, \Delta, F, \Gamma, \gamma)$ where the components have the following meaning: Q is a finite set of states, Σ is the input alphabet, $Q_I \subseteq Q$ is a set of initial states, $\Delta \subseteq Q \times \Sigma \times Q \times Q$ is the transition relation, $F \subseteq \Sigma \times Q$ the set of final letter/state combinations, Γ the finite set of counters and $\gamma : Q \to (\Gamma \to \{\mathtt{i}, \mathtt{r}, \mathtt{cr}\}^*)$ the counter actions. A run ρ of $\mathcal{A}$ on a tree t is also a tree with $\mathrm{dom}(\rho) = \mathrm{dom}(t)$, but with a labeling from Q that is consistent with the transition relation Δ. We call ρ accepting if $\rho(\varepsilon) \in Q_I$ and there are matching letter/states pairs for the leaves in F. Formally, for all $u \in \mathrm{leaves}(t) : (t(u), \rho(u)) \in F$. In the same way as for cost automata on words, we define B- and S-tree automata. They inherit the restrictions on the counter operations from word automata and their quantitative semantics is defined in the same spirit. However, we now compute the value along all paths from the root to a leaf in the tree as for word automata and take the maximum of the values in B-automata and the minimum in S-automata. Again, we write $[\![\mathcal{A}]\!]_B$ and $[\![\mathcal{A}]\!]_S$ to refer to this semantics.

For an analysis of tree automata, it is often helpful to take a game-theoretic viewpoint to the membership problem. To this end, we define *cost games* following the idea of [12]. For our purpose, it is helpful to view a cost game as a standard two-player reachability game on a finite graph that is extended with a finite set Γ of counters and counter actions γ that map every position in the game to the counter actions as for cost tree automata. As usual, the game positions are partitioned into two sets: One that belongs to the first player – called Eve – and one that belongs to the second player – called Adam. A play is formed similar to standard games: The play starts in some initial position. Then, the player that controls the respective game position chooses a next position according to the edges of the graph. Additionally, we simulate the counters along with the play and write $C(\tau)$ for the set of checked counter values in a play analogously to cost automata. For an introduction to games and their connection to tree automata see, e.g., [20].

We also define B- and S-games with the restrictions on the counter actions as for automata. In both types of games, Eve aims to reach the goal positions F of the reachability game. In B-games she additionally wants to minimize the largest checked counter value. In S-games she wants to maximize the smallest checked counter values. This is analogous to

automata. Correspondingly, an infinite play that never reaches F has value ∞ in B-games and value 0 in S-games. We define the *value* of a game based on the best values the players can enforce based on their *strategies*. For this purpose, we consider strategies as in standard two-player games on graphs. If Eve and Adam fix their strategies σ_E and σ_A, the resulting play is fixed and we denote it by τ_{σ_E,σ_A}. Cost games are determined (see [12]). That is, $\inf_{\sigma_E}\sup_{\sigma_A}\mathrm{val}_B(\tau_{\sigma_E,\sigma_A}) = \sup_{\sigma_A}\inf_{\sigma_E}\mathrm{val}_B(\tau_{\sigma_E,\sigma_A})$ and $\sup_{\sigma_E}\inf_{\sigma_A}\mathrm{val}_S(\tau_{\sigma_E,\sigma_A}) = \inf_{\sigma_A}\sup_{\sigma_E}\mathrm{val}_S(\tau_{\sigma_E,\sigma_A})$ and we write $\mathrm{val}_B(\mathcal{G})$ and $\mathrm{val}_S(\mathcal{G})$ for this value of the game $\mathcal{G}$.

It is helpful for our analysis to view the value computation of a tree t on a cost automaton as a cost game. The idea is essentially identical to the *membership game* (see [20]) for tree automata. The two players partly construct a run on one path of the tree from the root to a leaf. It starts in the root with some state from Q_I. In every position, Eve selects the transition from Δ that should be used and Adam chooses whether he wants to continue in the left or right child of the current node. The game continues in this node with the state given by the chosen transition. The counters and counter actions are just copied from the respective states in the automaton. The final positions are determined by the final letter/state pairs. We call this game the *cost membership game* of $\mathcal{A}$ on t and write $\mathcal{M}_{\mathcal{A},t}$. We have that $\mathrm{val}_B(\mathcal{M}_{\mathcal{A},t}) = [\![\mathcal{A}]\!]_B(t)$ and $\mathrm{val}_S(\mathcal{M}_{\mathcal{A},t}) = [\![\mathcal{A}]\!]_S(t)$.

3 Quantitative Counting MSO

In this section we introduce a quantitative extension of MSO with a focus on counting. We do so following the approach used for the quantitative μ-calculus [14]. Thus, we keep the standard syntax, and also use the traditional interpretations of the operators. However, instead of interpreting these over $\{0,1\}$, we evaluate minimums and maximums over the natural numbers with infinity.

To be more precise, we start with MSO without first-order variables and without negation. We then add a new atom to count the sizes of sets to the syntax, and allow formulas to be compared to infinity. Fixing a relational signature $\tau = \{R_1,\dots,R_n\}$ and a set $\mathcal{V} = \{X_1,\dots,X_m\}$ of second-order variables, formulas of qcMSO are defined inductively.

- Atomic formulas are of the form $RX_1\dots X_r$, $X_i = \varnothing$, $X_i \in X_j$, or $|X_i|$, for variables X_k and relation symbols $R \in \tau$ of arity r.
- If φ,ψ are formulas, then $\varphi\wedge\psi$, $\varphi\vee\psi$, $\varphi < \infty$, and $\varphi = \infty$ are formulas.
- If φ is a formula and X a variable, then $\exists X\varphi$ and $\forall X\varphi$ are formulas.

In contrast to the logics defined before, we view ∞ as true and 0 as false, which allows us to adapt the classical semantics. Furthermore, all atomic formulas but formulas $|X|$ are boolean. Given a τ-structure $\mathfrak{A}$ and an interpretation $\beta\colon X \to \mathcal{P}(A)$, the semantics $[\![\cdot]\!]^{\mathfrak{A},\beta}\colon \mathrm{qcMSO}(\tau) \to \mathbb{N}\cup\{\infty\}$ is defined as follows, where we omit $\mathfrak{A},\beta$ if clear from the context for better readability.

$$[\![|X|]\!] = |\beta(X)| \qquad [\![RX_1\dots X_r]\!] = \begin{cases}\infty, & (a_1,\dots,a_r)\in R, \beta(X_i)=\{a_i\}\\ 0, & \text{otherwise}\end{cases}$$

$$[\![X=\varnothing]\!] = \begin{cases}\infty, & \beta(X)=\varnothing\\ 0, & \text{otherwise}\end{cases} \qquad [\![X\in Y]\!] = \begin{cases}\infty, & \beta(X)=\{a\}, a\in\beta(Y)\\ 0, & \text{otherwise}\end{cases}$$

$$[\![\varphi\vee\psi]\!] = \max([\![\varphi]\!],[\![\psi]\!]) \qquad [\![\varphi\wedge\psi]\!] = \min([\![\varphi]\!],[\![\psi]\!])$$

$$[\![\varphi<\infty]\!] = \begin{cases}\infty, & [\![\varphi]\!]<\infty\\ 0, & \text{otherwise}\end{cases} \qquad [\![\varphi=\infty]\!] = \begin{cases}\infty, & [\![\varphi]\!]=\infty\\ 0, & \text{otherwise}\end{cases}$$

$$[\![\exists X\varphi]\!] = \sup_{A'\subseteq A}[\![\varphi]\!]^{\beta[X\mapsto A']} \qquad [\![\forall X\varphi]\!] = \inf_{A'\subseteq A}[\![\varphi]\!]^{\beta[X\mapsto A']}$$

As a convention, formulas whose evaluation can only be 0 or ∞, that is, `true` or `false`, are called boolean formulas. For example, formulas without any occurrence of $|X|$ are boolean, and so are formulas $\varphi = \infty$.

As atomic formulas which are also formulas of MSO always evaluate to ∞ and 0, qcMSO clearly extends MSO: a formula can be translated by replacing every occurrence of a negation with a comparison $< \infty$. What is more, it is easily expressible that a set is finite. As supremums and infimums over sets that satisfy a certain property can be encoded in a straightforward manner, we can thus also express the quantifier U:

$$UX.\varphi \equiv (\exists X(|X| \wedge |X| < \infty \wedge \varphi)) = \infty.$$

It is an easy consequence that qcMSO subsumes MSO+U.

▶ **Lemma 2.** *For every formula $\varphi \in$ (W)MSO+U there is a formula $\varphi' \in$ qc(W)MSO such that $\mathfrak{A} \vDash \varphi$ if and only if $[\![\varphi']\!]^{\mathfrak{A}} = \infty$.*

One easily obtains the following lemma by exchanging $\wedge$ and $\vee$, swapping the quantifiers, replacing positive boolean atomic formulas φ by $\varphi < \infty$ and negated atoms $\neg\varphi$ by φ to take the inverted semantics between costMSO and qc(W)MSO into account. Additionally, we need to rewrite $|X| < N$ to $|X|$ because of the different syntax of the operators.

▶ **Lemma 3.** *For every cost(W)MSO-formula φ there exists a qc(W)MSO-formula φ' such that $[\![\varphi]\!]^{\mathfrak{A}} = [\![\varphi']\!]^{\mathfrak{A}}$.*

Furthermore, the boundedness property and the dominance relation for costMSO on finite words and trees are expressible in qcMSO on $(\omega, <)$ and the infinite binary tree $\mathfrak{T}_2 = (\{0,1\}^*, S_0, S_1)$, respectively, as stated by the following lemma.

▶ **Lemma 4.**

1. *Given a costMSO-formula φ over finite words, one can effectively construct a qcWMSO-formula φ_b such that $[\![\varphi_b]\!]^{(\omega,<)} = \infty$ (`true`) if and only if φ is bounded over finite words.*
2. *Given two costMSO-formulas φ, ψ over finite words, one can effectively construct a qcMSO-formula ϑ_d such that $[\![\vartheta_d]\!]^{\mathfrak{T}_2} = \infty$ if and only if φ dominates ψ on finite words.*

Proof. Regarding 1., a finite word over $\Sigma = \{0,1\}$ can be represented by two sets X, X_1 such that $X \subseteq \omega$ is a finite initial subset of the natural numbers indicating the length of the word and $X_1 \subseteq X$ (also finite) contains the positions labeled with 1. Clearly, such sets can be defined in WMSO and thus qcWMSO by a formula $\psi(X, X_1)$. As costMSO is subsumed by qcMSO, there exists a qcMSO-formula φ' with the same evaluation. Let φ'_r be this formula where quantification is relativized to (finite) X and $P_1 Y$ is replaced by $Y \in X_1$. Then, boundedness is expressed by $(\exists X \exists X_1(\psi(X, X_1) \wedge \varphi'_r)) < \infty$.

Regarding 2., we identify a word $w \in \{0,1\}^*$ with the respective position in the tree. For MSO, it is easy to define, given a sentence φ over finite words, a formula $\varphi'(x)$ over the infinite binary tree such that $w \vDash \varphi$ if and only if $\mathfrak{T}_2 \vDash \varphi'(w)$. This directly extends to costMSO, and accordingly, we also obtain corresponding translations $\widehat{\varphi}(X), \widehat{\psi}(X) \in$ qcMSO for φ, ψ. Dominance is then expressed by $\forall X[(\exists Y(Y \in X \wedge \widehat{\varphi}(Y))) = \infty \vee (\exists Y(Y \in X \wedge \widehat{\psi}(Y))) < \infty]$. We remark that it is important here that $\forall X$ also quantifies over infinite sets to match the definition of dominance. ◀

By adapting the construction for the boundedness formula, it follows straightforwardly that boundedness on infinite words and finite trees can also be expressed (in the respective structures).

▶ **Corollary 5.**

- *The boundedness problem for costMSO on infinite words can be expressed in qcMSO on* $(\omega, <)$.
- *The boundedness problem for costMSO on finite trees can be expressed in qcWMSO on the infinite binary tree.*

As qcMSO extends both costMSO and MSO+U, negative results for either of these transfer. In fact, the conditional undecidability result for MSO+U on the infinite binary tree from [5] was recently improved: it was shown in [6] that the MSO+U-theory of $(\omega, <)$ is undecidable. Thus, the undecidability of the model-checking problem for qcMSO follows:

▶ **Corollary 6.** *Given a qcMSO-sentence* φ*, it is undecidable whether* $[\![\varphi]\!]^{(\omega,<)} = \infty$.

4 Resource-Automatic Structures and FO+RR$^{=\infty}$

Towards the decidability of the qcWMSO-theories of the natural numbers with order and the infinite binary tree, we follow an approach used in [19]: Instead of working with second-order quantification directly, we consider the first-order problem on the powerset structure restricted to finite sets. We prove that the respective structures are resource-automatic and resource-tree-automatic, respectively, and that deciding the theory reduces to deciding the theory of an extension of FO+RR with infinity comparisons.

As a first step, we extend syntax and semantics of FO+RR introduced earlier by the new operators $\varphi = \infty$ and $\varphi < \infty$ for formulas φ such that $\varphi = \infty$ evaluates to 0 if $[\![\varphi]\!] = \infty$ and to ∞ otherwise, and dually for $\varphi < \infty$. We use the decidability of FO+RR established in [18], which itself relies on the closure properties for regular cost functions described in [9]. To prove that this extension FO+RR$^{=\infty}$ is still decidable on resource-automatic structures, it suffices to provide an automata-theoretical construction that transforms a B-automaton $\mathcal{A}$ for φ into a B-automaton $\mathcal{A}_\infty$ for $\varphi = \infty$.

▶ **Lemma 7.** *Let* $\mathcal{A}$ *be a* B*-automaton. One can effectively construct a* B*-automaton* $\mathcal{A}_\infty$ *such that* $[\![\mathcal{A}_\infty]\!](w) = 0$ *if* $[\![\mathcal{A}]\!](w) = \infty$ *and* $[\![\mathcal{A}_\infty]\!](w) = \infty$ *otherwise*

Proof. As we consider automata over finite words, observe that $[\![\mathcal{A}]\!](w) = \infty$ only if there are no accepting runs of $\mathcal{A}$ on w. Hence, to construct $\mathcal{A}_\infty$, we first view $\mathcal{A}$ as an NFA by removing all counters, then complement it, and reintroduce a dummy counter that is never checked. By construction, this automaton $\mathcal{A}_\infty$ maps a word to 0 if and only if $\mathcal{A}$ as an NFA rejects the word, and all other words are mapped to ∞. ◀

▶ **Corollary 8.** *Given an FO+RR*$^{=\infty}$*-sentence* φ *and a resource-automatic structure* $\mathfrak{A}$*, it is decidable whether* $[\![\varphi]\!]^{\mathfrak{A}} = \infty$.

To prove that the qcWMSO-theory of $(\omega, <)$ is decidable, consider the structure $\mathfrak{F} = (\mathrm{FinPot}(\mathbb{N}), <, |\cdot|, \in, = \varnothing)$, where

- $\mathrm{FinPot}(\mathbb{N}) = \{a \subseteq \mathbb{N} \mid |a| < \infty\}$,
- $a < b = \infty$ if $a = \{a'\}, b = \{b'\}$ are singleton sets such that $a' < b'$, and $a < b = 0$ otherwise,
- $|a|$ evaluates to the size of a,
- $a \in b = \infty$ if $a = \{a'\}$ is a singleton and $a' \in b$, and $a \in b = 0$ otherwise,
- and $a = \varnothing$ evaluates to ∞ if a is indeed empty and to 0 otherwise.

It is not difficult to see that this structure is resource-automatic, using the regular language $\{0\} \cup \{0,1\}^*1$ where a word corresponds to the set that consists of the indices where the

word is 1. For $|\cdot|$, a single counter to count the occurrences of 1s suffices. The other relations correspond to the complements of the relations in the classical automatic structures setting.

Following the approach used to embed costMSO into qcMSO, by exchanging conjunctions and disjunctions, replacing $\exists X$ by $\forall x$ and $\forall X$ by $\exists x$ and swapping $= \infty$ and $< \infty$, one obtains the following lemma.

▶ **Lemma 9.** *For every qcWMSO-sentence φ, one can effectively construct an FO+RR$^{=\infty}$-sentence φ' such that $[\![\varphi]\!]^{(\omega,<)} = [\![\varphi']\!]^{\mathfrak{F}}$.*

▶ **Corollary 10.** *Given a qcWMSO-sentence φ, it is decidable whether $[\![\varphi]\!]^{(\omega,<)} = \infty$.*

It was argued in [18] that exact evaluations of FO+RR-sentences on resource-automatic structures can be computed once it is known that the evaluation is bounded. This can, for example, be achieved by successively trying parameters $n = 0, 1, \ldots$ with standard first-order evaluation on the structure where a resource relation R is replaced by the relation of tuples of R of cost at most n. As the automata $\mathcal{A}_\infty$ have boolean evaluations, thus are essentially NFAs, this approach can be lifted to FO+RR$^{=\infty}$-sentences, by substituting only relations outside the scope of ∞-comparisons. Accordingly, exact evaluations can be computed also for qcWMSO on the ordered natural numbers.

▶ **Theorem 1.**

(a) *Given a sentence $\varphi \in$ qcWMSO, the evaluation $[\![\varphi]\!]^{(\omega,<)}$ of φ on the natural numbers with order is effectively computable.*

4.1 Resource-Tree-Automatic Structures

We aim at generalizing the idea of resource-automatic structures to universes that are representable as a regular tree language. This extends the well-understood idea of tree-automatic structures (see, e.g., [1]). Moreover, it allows us to reuse the idea presented previously to obtain an algorithm for qcWMSO on the infinite binary tree, because finite trees can be used to represent all finite subsets of the infinite binary tree.

The general approach to resource-tree-automatic structures and the presentation of the result follow the ideas for resource-automatic structures as presented in [18]. First, we fix some additional notation. Secondly, we describe an inductive translation strategy from FO+RR$^{=\infty}$-formulas to cost tree automata. While the cases of the boolean connectives can be directly transferred, the translation of the quantifiers needs some more insight into cost games. Correspondingly, we analyze strategies in S-games and use the results to complete the inductive translation, which provides an algorithmic method to compute the (approximate) value of FO+RR$^{=\infty}$-formulas.

First, we extend the definition of convolution and transducers to trees. For a finite alphabet Σ, let $\Sigma^{\otimes m} = (\Sigma \cup \{\$\})^m$ and let $t_1 \in \mathcal{T}_{\Sigma^{\otimes m}}, t_2 \in \mathcal{T}_\Sigma$ be two trees. We define the *convolution* by $t := t_1 \otimes t_2 \in \mathcal{T}_{\Sigma^{\otimes m+1}}$ with $\mathrm{dom}(t) = \mathrm{dom}(t_1) \cup \mathrm{dom}(t_2)$ and $t(u)_i = t_1(u)_i$ if $u \in \mathrm{dom}(t_1)$, $t(u)_i = \$$ otherwise for $i \leq m$ and $t(u)_{m+1} = t_2(u)$ if $u \in \mathrm{dom}(t_2)$, $t(u)_{m+1} = \$$ otherwise. A tree $t \in \mathcal{T}_{\Sigma^{\otimes m}}$ is *correctly padded* if there are $t_1, \ldots, t_m \in \mathcal{T}_\Sigma$ such that $t = t_1 \otimes \cdots \otimes t_m$. This means correctly padded trees have have no \$ if there are normal letters in the same component of a descendant. Moreover, they have no positions labeled completely with padding symbols. We write $\square := \$^m$ as a short-hand for a vector of appropriate dimensionality containing only padding symbols. An m-dimensional synchronous B-/S-tree transducer $\mathcal{A}$ is a cost tree automaton operating over the alphabet $\Sigma^{\otimes m}$. We define its semantics by the cost automaton semantics over the convolution: $[\![\mathcal{A}]\!] : (\mathcal{T}_\Sigma)^m \to \mathbb{N} \cup \{\infty\}$, $(t_1, \ldots, t_m) \mapsto [\![\mathcal{A}]\!]_{B/S}(t_1 \otimes \cdots \otimes t_m)$. With these preparations, we can define resource-tree-automatic structures.

▶ **Definition 11.** Let $\mathfrak{S} = (S, R_1, \ldots, R_m)$ be a resource structure. We call $\mathfrak{S}$ resource-tree-automatic if S is representable as a regular language of finite trees and there are synchronous B-/S-tree transducers $\mathcal{A}_{R_i}$ such that $R_i^{\mathfrak{S}} = [\![\mathcal{A}_{R_i}]\!]$.

For the purpose of a clearer proof presentation we will assume that $S = \mathcal{T}_\Sigma$. This is no restriction of the general case since we can introduce a new automatic predicate $P_S \subseteq \mathcal{T}_\Sigma$ that contains exactly the elements of S and relativize all quantifications w.r.t. P_S for every regular tree language S.

4.1.1 Translating FO+RR$^{=\infty}$ to transducers

We now provide the necessary ingredients for an inductive translation of FO+RR$^{=\infty}$-formulas into synchronous cost transducers. For a given formula φ with k free variables, we construct a k-dimensional synchronous cost transducer $\mathcal{A}_\varphi$ such that $[\![\varphi]\!] = [\![\mathcal{A}_\varphi]\!]$. The cases of atomic formulas are simple. For a relation R_i, we are given a cost transducer $\mathcal{A}_{R_i}$ by definition. The operators $=$ and $\neq$ can be implemented with simple cost tree automata that just check whether in all positions in the tree all letters in the alphabet vector match or that there is a mismatch somewhere, respectively. The semantics of the boolean connectives is min and max. Correspondingly, we directly use the closure of cost tree automata under min and max, which was already established in [12]. The $= \infty$ operator can be translated in the same way as for finite words: By the definition of the semantics of B-tree automata, the value of a tree is ∞ if and only if the tree is rejected in the classical sense, i.e., there is no strategy of Eve in $\mathcal{M}_{\mathcal{A},t}$ to always reach a final state in every play. Thus, we can obtain an automaton for $= \infty$ by ignoring the counters and constructing a classical complement automaton for the given one. When interpreted as a B-tree automaton, it will output 0 for trees that were previously not accepted (that is, had value ∞) and ∞ otherwise.

It remains to consider existential and universal quantification. In the classical setting, universal quantification can be expressed by negation and existential quantification. In our setting, we have to consider both existential and universal quantification since FO+RR$^{=\infty}$ has no negation. We deal with these quantifications by inf-projection and sup-projection for cost automata. However, on the automaton level, one has to deal with the padding symbol. After projecting away one of the components, the automaton may contain transitions only labeled with padding symbols. Let us illustrate the problem by a simple example over $\Sigma = \{a, b\}$: Let Rxy be a binary relation symbol that evaluates to the length of the longest path in y. Formally, $R(t_1, t_2) := \max_{u \in \mathrm{dom}(t_2)} |u|$. A B-tree automaton $\mathcal{A}$ that just increments its counter in every step on the second tree implements this relation. Now, consider the tree $t_0 = a$ that consists only of a root node. For this tree, we have $[\![\forall y R t_0 y]\!] = \infty$ but if we look at the sup-projection of $\mathcal{A}$ to the first component denoted by $\mathcal{A}^{\mathrm{sup}}$ we obtain

$$[\![\mathcal{A}^{\mathrm{sup}}]\!](a) = \max\{[\![\mathcal{A}]\!]((a,a)), [\![\mathcal{A}]\!]((a,b)), [\![\mathcal{A}]\!]((a,\$))\} = 1$$

To compute the supremum over all trees, we have to consider t_0 extended by arbitrarily long sequences of padding symbols ($\square$).

In the classical setting (for existential quantification and standard projection), this can be handled by treating such pure padding transitions as ε-transitions and then eliminating them. However, in cost automata the pure padding transitions are much more difficult to eliminate (see [18] where this problem is treated for cost automata on words). As a solution, we split the computation of sup and inf into the respective projection operation and an

additional sup/inf over arbitrarily long padding sequences. Formally, for a tree $t \in \mathcal{T}_{\Sigma^{\otimes m}}$, let

$$\operatorname{padext}(t) := \left\{ s \in \mathcal{T}_{\Sigma^{\otimes m}} \mid \operatorname{dom}(s) \supseteq \operatorname{dom}(t), s(u) = \begin{cases} t(u), & u \in \operatorname{dom}(t) \\ \square, & \text{otherwise} \end{cases} \right\}$$

Consider the case of existential quantification, and assume that we have applied the inf-projection and obtained a B-automaton $\mathcal{A}$ (which might still contain pure padding transitions). For a tree $t \in \mathcal{T}_{\Sigma^{\otimes m}}$ we are interested in the value $\inf_{s \in \operatorname{padext}(t)} [\![\mathcal{A}]\!]_B(s)$. If we modify $\mathcal{A}$ such that it can simulate on t all runs of $\mathcal{A}$ on trees from $\operatorname{padext}(t)$, then we obtain the correct value for t because the semantics of B-automata takes the infimum over all runs. Similarly, we use S-automata for the sup-projection to obtain $\sup_{s \in \operatorname{padext}(t)} [\![\mathcal{A}]\!]_S(s)$.

Thus, for a given tree t and a cost tree automaton $\mathcal{A}$, we aim at modifying $\mathcal{A}$ such that it can simulate all runs on trees from $\operatorname{padext}(t)$ that contribute to the overall value. For this purpose, we consider the membership game on the infinite tree $t_\square$ that is labeled with $\square$ at all positions. Since all positions in this tree look the same, there is no need to keep track of the position in the tree. Moreover, Eve decides in each round whether the current node is treated as an inner node or as a leaf (since Eve has to reach the goal set, she has to decide for a leaf at some point). We call this the *padding game* for $\mathcal{A}$ and denote it by $\mathcal{M}_{\mathcal{A},\square}$. We now investigate the "concatenation" of the cost membership game followed by the padding game.

Let $(\mathcal{M}_{\mathcal{A},t} \triangleright \mathcal{M}_{\mathcal{A},\square})$ consist of the union of $\mathcal{M}_{\mathcal{A},t}$ and $\mathcal{M}_{\mathcal{A},\square}$ with the following additional connecting edges: from a position (q,u) of $\mathcal{M}_{\mathcal{A},t}$ with a leaf u of t, Eve can decide to stay inside $\mathcal{M}_{\mathcal{A},t}$ and finish the game as usual, or to treat u as an inner node. In the latter case, the play would reach a position (q', ui) for a node ui not in the domain of t. Instead, the play jumps to $\mathcal{M}_{\mathcal{A},\square}$ in state q'.

▶ **Lemma 12.** *Let $\mathcal{A}$ be a nondeterministic cost tree automaton. We have:*

- $\operatorname{val}_B((\mathcal{M}_{\mathcal{A},t} \triangleright \mathcal{M}_{\mathcal{A},\square})) = \inf_{s \in \operatorname{padext}(t)} [\![\mathcal{A}]\!]_B(s)$
- $\operatorname{val}_S((\mathcal{M}_{\mathcal{A},t} \triangleright \mathcal{M}_{\mathcal{A},\square})) = \sup_{s \in \operatorname{padext}(t)} [\![\mathcal{A}]\!]_S(s)$

With this knowledge and the observation that $\mathcal{M}_{\mathcal{A},\square}$ does not depend on the input tree t, we develop methods to precompute information on $\mathcal{M}_{\mathcal{A},\square}$ with the goal of providing a modified automaton $\mathcal{A}'$ whose membership game $\mathcal{M}_{\mathcal{A}',t}$ approximates (in the sense of $\approx$ equivalence) the combined game $(\mathcal{M}_{\mathcal{A},t} \triangleright \mathcal{M}_{\mathcal{A},\square})$.

First, we consider the case of inf. This case is easier due to the inherent asymmetry in our setting. We claim that it is sufficient to know from which positions $(q,\square)$ Eve can win $\mathcal{M}_{\mathcal{A},\square}$ when we interpret this as a simple reachability game with goal set F. This has the following justification: In B-games every counter is always checked after an increment. Thus, the value of a play can only increase and Eve should just reach a final position as fast as possible. If she just plays the normal reachability strategy she reaches F in at most as many steps as the size of $\mathcal{M}_{\mathcal{A},\square}$ (denoted by $|\mathcal{M}_{\mathcal{A},\square}|$). In the worst case, every of these steps increments the counter. However, even if she could have avoided some of these increments, the error w.r.t. an optimal strategy is at most $|\mathcal{M}_{\mathcal{A},\square}|$. Thus, we obtain:

▶ **Lemma 13.** *Let $\mathcal{A}$ be a synchronous B-tree transducer over $\Sigma^{\otimes m}$. One can construct a synchronous B-tree transducer $\mathcal{A}'$ such that: $[\![\mathcal{A}']\!]_B(t) \approx \inf_{s \in \operatorname{padext}(t)} [\![\mathcal{A}]\!]_B(s)$.*

The sup-case requires more sophisticated methods for two major reasons: First, the independent use of increment and check prevent an argument as before. Secondly, there is no single strategy witnessing $\operatorname{val}_S(\mathcal{G}) = \infty$. Moreover, we recognize that it does not suffice to compute the value of $\mathcal{M}_{\mathcal{A},\square}$ to capture the behavior of $(\mathcal{M}_{\mathcal{A},t} \triangleright \mathcal{M}_{\mathcal{A},\square})$ in an automaton. In the combined game, the counters are not initialized with 0 at the beginning of the $\mathcal{M}_{\mathcal{A},\square}$-part

but inherit the current counter values from $\mathcal{M}_{\mathcal{A},t}$. So a direct $\mathtt{cr}$ at the beginning of $\mathcal{M}_{\mathcal{A},\Box}$ ensures that Adam can always achieve value 0 if $\mathcal{M}_{\mathcal{A},\Box}$ is considered individually. But in the combined game the counter may have a large value from $\mathcal{M}_{\mathcal{A},t}$. If he could first reset the counter before checking it, this would be much better.

We approach this problem by a more detailed analysis of strategies of Adam in S-cost games. Due to space restrictions, we can only provide the cornerstones of the proof strategy here. Let σ be a strategy of Adam in an S-cost game $\mathcal{G}$. We remind the reader that Adam wants to check counters with *small* values or avoid F. Since we can handle the reachability of F individually before, we concentrate on the counter values. We measure the success of σ in the following three categories per counter:

1. $\mathtt{cr}$: The strategy σ can enforce a check of the counter after a bounded number of steps.
2. $\mathtt{rcr}$: The strategy σ can enforce a reset before any check and subsequently a check after a bounded number of increments
3. $\bot$: The strategy σ provides no guarantees on the counter.

Formally, such a profile p is a mapping from the set of counters Γ to $\{\mathtt{cr}, \mathtt{rcr}, \bot\}$ and we say that a strategy σ guarantees p if all plays that are played according to σ satisfy the conditions stated in p. We call a profile p *simple* and write, e.g., $[c_1 \mapsto \mathtt{cr}]$ if it provides only a guarantee on one counter – the other counters are mapped to $\bot$. In order to incorporate the choices of Eve in the play, we need combinations of several such profiles to describe a strategy σ. For example, Eve may choose whether she wants to check a counter c_1 or c_2. We express this as a disjunction of profiles (called generalized profile) and write in this example $[c_1 \mapsto \mathtt{cr}] \vee [c_2 \mapsto \mathtt{cr}]$.

Computing all possible generalized profiles for strategies of Adam in an S-cost game is a sufficient precomputation to approximate the complete game $(\mathcal{M}_{\mathcal{A},t} \triangleright \mathcal{M}_{\mathcal{A},\Box})$ at the position of a leaf node in $\mathcal{M}_{\mathcal{A},t}$. There are only finitely many generalized profiles. We want to compute for every such generalized profile whether Adam has a strategy that guarantees it. We then want to simulate the behavior of $(\mathcal{M}_{\mathcal{A},t} \triangleright \mathcal{M}_{\mathcal{A},\Box})$ in an automaton. To implement this, we use the fact that one can extend nondeterministic cost tree automata to alternating cost tree automata in a similar way as standard alternating tree automata (see [12]). The alternation allows us to represent Adam's choice among his possible generalized profiles followed by Eve's choice among the profiles in the disjunction in the automaton. In the target state, the automaton checks counters that have guarantee $\mathtt{cr}$ and resets and then checks counters that have guarantee $\mathtt{rcr}$. A detailed analysis of S-cost games shows that the bound on the number of increments in the guarantees $\mathtt{cr}$ and $\mathtt{rcr}$ only depends on the size of the game. Since the size of $\mathcal{M}_{\mathcal{A},\Box}$ only depends on the size of $\mathcal{A}$, we obtain the following:

▶ **Lemma 14.** *Let $\mathcal{A}$ be a synchronous S-tree transducer over $\Sigma^{\otimes m}$. One can construct a synchronous S-tree transducer $\mathcal{A}'$ such that: $[\![\mathcal{A}']\!]_S(t) \approx \sup_{s \in \mathrm{padext}(t)} [\![\mathcal{A}]\!]_S(s)$.*

The algorithm to compute the possible generalized profiles employs methods from the theory of tree automata on infinite trees. We unfold plays on a finite game graph as an infinite tree and notice that we can approximate strategies that are good for Adam (as they guarantee $\mathtt{cr}$ or $\mathtt{rcr}$) in MSO logic. From the fact that MSO-formulas always have a regular tree as model, we can deduce a bound on the number of increments.

In a last step, we combine all the previous observations and results from the theory of regular cost functions over trees. We saw how to inductively transform an FO+RR$^{=\infty}$-formula φ over a resource-tree-automatic structure into a synchronous cost tree transducer that computes the semantics up to $\approx$. The changes from S- to B-automata (or vice versa) and from alternating to nondeterministic can also be computed effectively up to $\approx$ (cf. [12]). In

total, we obtain a transducer $\mathcal{A}$ such that $[\![\mathcal{A}]\!] \approx [\![\varphi]\!]$. If the computed value is ∞, this is even exact. In the other case, we know that $[\![\varphi]\!]$ has a finite value. As described earlier for the case of resource-automatic structures on words, we can similarly compute exact values by a reduction to standard tree-automatic structures for formulas outside of ∞-comparisons.

▶ **Theorem 15.** *The semantics of FO+RR$^{=\infty}$ is effectively computable on resource-tree-automatic structures.*

4.1.2 Deciding the qcWMSO-theory of $\mathfrak{T}_2$

It remains to show that the above result can be applied to establish that the qcWMSO-theory of the infinite binary tree is decidable. We use the same idea as for $(\omega, <)$ to obtain this result. Given the infinite binary tree $\mathfrak{T}_2 = (\{0,1\}^*, S_0, S_1)$, we consider the following variant of the finite powerset structure $\mathfrak{F}_2 = (\mathrm{FinPot}(\{0,1\}^*), S_0, S_1, \rangle\cdot\langle, \in, = \varnothing)$, where the (resource) relations are as follows:

- $(a,b) \in S_i$ evaluates to ∞ if both $a = \{a'\}, b = \{b'\}$ are singletons and $b' = a'i$. Otherwise, it evaluates to 0.
- For a set $a \subseteq \{0,1\}^*$, $\rangle a \langle$ evaluates to an approximation of the cardinality of a that can easily be computed by a cost automaton (see below for a further explanation), and is defined as follows:

$$\rangle a \langle = \sup_{w \in a} \left(|\{i \mid \exists u \in a : w[1 \dots i] \preceq u \wedge w[1 \dots i+1] \not\preceq u\}| \right).$$

- $a \in b$ evaluates to ∞ if $a = \{a'\}$ is a singleton such that $a' \in b$, and to 0 otherwise.
- $a = \varnothing$ evaluates to ∞ if a is empty, and to 0 otherwise.

Note that $\rangle\cdot\langle$ in the above structure is different from the evaluation of $|\cdot|$ in qcWMSO. However, boundedness is preserved because we have $\rangle a \langle \leq [\![|a|]\!]^{\mathfrak{T}_2} \leq 2^{\rangle a \langle}$: The first inequality comes from the fact that each element of a can contribute at most 1 to the value of $\rangle a \langle$. To see the second one, consider the following path w for the supremum: Always proceed to the subtree that contains more than half of the remaining elements of a. This counts one if the non-selected subtree was not empty but loses at most half of the remaining elements.

We now claim that $\mathfrak{F}_2$ is resource-tree-automatic. As universe we consider the set $S \subseteq \mathcal{T}_{\{0,1\}}$ of finite binary trees with the property that every inner node of the tree is either labeled with 1 or has a 1-labeled node as descendant. This property can easily be checked by a tree automaton. A labeled tree t corresponds to the subset of $\mathfrak{T}_2$ of those positions in t that are labeled with 1. The condition on the trees in the universe ensures a unique encoding of every set. Clearly, S_0, S_1 are automatic, and so are $\in$ and $= \varnothing$, as they are tree-automatic in the classical sense. The relation $\rangle\cdot\langle$ can be implemented with an S-automaton that simulates walks as described above by nondeterministically guessing and verifying the positions where the not-selected subtree contains elements of a.

Using the same translation as for ω, we reduce the problem of deciding the theory of qcWMSO to the boundedness problem for FO+RR$^{=\infty}$. This is possible as $\rangle\cdot\langle$ in the above sense preserves boundedness.

▶ **Theorem 1.**
(b) *Let φ be a qcWMSO-sentence. It is decidable whether $[\![\varphi]\!]^{\mathfrak{T}_2} = \infty$.*

Although exact FO+RR$^{=\infty}$-evaluations can be computed on resource-tree-automatic structures, this does not entail that qcWMSO can be evaluated exactly on $\mathfrak{T}_2$. This shortcoming has its origin in the fact that cost tree automata can count only along paths. To have an exact reduction from qcWMSO to FO+RR$^{=\infty}$, counts of different paths would have to be combined to simulate counting the size of an arbitrary subset of $\{0,1\}^*$.

5 Conclusion

We introduced the logic qcMSO as a quantitative MSO variant to specify boundedness properties. The logics costMSO and MSO+U can be embedded into qcMSO in a natural way. Thus, the undecidability of MSO+U on $(\omega, <)$ already shows that the semantics of qcMSO on $(\omega, <)$ and the infinite binary tree $\mathfrak{T}_2$ is not computable. Accordingly, we focused on the weak variant qcWMSO and provide a method to compute the value of qcWMSO sentences on $(\omega, <)$ and approximations of the value on $\mathfrak{T}_2$. This result is achieved by a reduction to a cost-function extension of automatic structures – called resource-automatic structures. Moreover, we lift the known results to resource-tree-automatic structures.

In the future, we would like to see whether there is an automaton model for qcWMSO and whether there are meaningful fragments of full qcMSO with good algorithmic properties.

References

1 A. Blumensath. Automatic Structures. Diploma thesis, RWTH-Aachen, 1999.
2 A. Blumensath and E. Grädel. Automatic Structures. In *LICS 2000*, pages 51–62, 2000.
3 M. Bojanczyk. Weak MSO with the Unbounding Quantifier. In *STACS 09*, volume 3, pages 159–170, 2009.
4 M. Bojańczyk. Weak MSO with the unbounding quantifier. *Theory of Computing Systems*, 48(3):554–576, 2011.
5 M. Bojańczyk, T. Gogacz, H. Michalewski, and M. Skrzypczak. On the decidability of MSO+U on infinite trees. In *Automata, Languages, and Programming*, volume 8573 of *LNCS*, pages 50–61. Springer, 2014.
6 M. Bojańczyk, P. Parys, and S. Toruńczyk. The MSO+U theory of $(\mathbb{N}, <)$ is undecidable. *arXiv:1502.04578 [cs.LO]*, 2015.
7 M. Bojanczyk and S. Torunczyk. Weak MSO+U over infinite trees. In *STACS 2012*, volume 14, pages 648–660, Dagstuhl, Germany, 2012.
8 M. Bojańczyk and S. Toruńczyk. Weak MSO+U over infinite trees. In Christoph Dürr and Thomas Wilke, editors, *STACS 2012*, volume 14, pages 648–660, 2012.
9 T. Colcombet. Regular cost functions over words. *Manuscript available online*, 2009.
10 T. Colcombet. The theory of stabilisation monoids and regular cost functions. In *Automata, Languages and Programming*, volume 5556 of *LNCS*, pages 139–150. Springer, 2009.
11 T. Colcombet. Regular cost functions, part I: logic and algebra over words. *Logical Methods in Computer Science*, 9(3), 2013.
12 T. Colcombet and C. Löding. Regular cost functions over finite trees. In *LICS 2010*, pages 70–79, July 2010.
13 H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available online: `http://www.grappa.univ-lille3.fr/tata`, 2007. release October, 12th 2007.
14 D. Fischer, E. Grädel, and Ł. Kaiser. Model Checking Games for the Quantitative mu-Calculus. *Theory Comput. Syst.*, 47(3):696–719, 2010.
15 S. Hummel and M. Skrzypczak. The topological complexity of MSO+U and related automata models. *Fundamenta Informaticae*, 119(1):87–111, 2012.
16 B. Khoussainov and A. Nerode. Automatic presentations of structures. In *Logic and Computational Complexity*, volume 960 of *LNCS*, pages 367–392. Springer, 1995.
17 D. Kuperberg and M. Vanden Boom. On the expressive power of cost logics over infinite words. In *ICALP 2012*, volume 7392 of *LNCS*, pages 287–298. Springer, 2012.
18 M. Lang and C. Löding. Modeling and verification of infinite systems with resources. *Logical Methods in Computer Science*, 9(4), 2013.

19 M. Lang, C. Löding, and A. Manuel. Definability and transformations for cost logics and automatic structures. In *MFCS 2014*, volume 8634 of *LNCS*, pages 390–401. Springer, 2014.

20 W. Thomas. Languages, automata, and logic. In *Handbook of Formal Language Theory*, volume III, pages 389–455. Springer, 1997.

21 M. Vanden Boom. Weak cost monadic logic over infinite trees. In *MFCS 2011*, volume 6907 of *LNCS*, pages 580–591. Springer, 2011.

Deciding the First Levels of the Modal μ Alternation Hierarchy by Formula Construction

Karoliina Lehtinen and Sandra Quickert

Laboratory for Foundations of Computer Science, University of Edinburgh
10 Crichton Street, Edinburgh, UK
M.K.Lehtinen@sms.ed.ac.uk, squicke1@inf.ed.ac.uk

Abstract

We construct, for any sentence Ψ of the modal μ calculus (L_μ), a derived sentence Ψ^{ML} in the modal fragment ML of L_μ and a sentence $\Psi^{\Pi_1^\mu}$ in the fragment Π_1^μ of L_μ without least fixpoints such that Ψ is equivalent to a formula in ML or Π_1^μ if and only if it is equivalent to Ψ^{ML} or $\Psi^{\Pi_1^\mu}$ respectively. The formula $\Psi^{\Sigma_1^\mu}$ such that Ψ is equivalent to $\Psi^{\Sigma_1^\mu}$ if and only if Ψ is semantically in the greatest-fixpoint free fragment Σ_1^μ is obtained by duality to $\Psi^{\Pi_1^\mu}$. This yields a new proof of decidability of the first levels of the modal μ alternation hierarchy. The blow-up incurred by turning Ψ into the modal formula Ψ^{ML} is shown to be necessary: there are ML formulas that can be expressed sub-exponentially more efficiently with the use of fixpoints. For Π_1^μ and Σ_1^μ however, as long as formulas are in guarded disjunctive form, the transformation into a syntactically Π_1^μ or Σ_1^μ does not increase the size of the formula.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases modal μ calculus, fixpoint logic, alternation hierarchy

Digital Object Identifier 10.4230/LIPIcs.CSL.2015.457

1 Introduction

The modal μ calculus (L_μ), a logic expressing properties of labelled transition systems, was first introduced by Kozen in 1983 [5]. Its popularity is due to its simple but productive syntax and appealing decidability: deciding satisfiability is EXPTIME-complete; model checking is in NP and conjectured to be in P.

Syntactically, L_μ consists simply of a propositional modal logic augmented with its namesake least fixpoint operator μ and the dual greatest fixpoint operator ν. Both the expressivity and complexity of the logic stem from the alternating usage of μ and ν: the more alternations are allowed, the richer the fragment of L_μ but the more difficult its model-checking. Indeed, the alternation hierarchy, consisting of L_μ fragments for which the number of alternations is fixed is strict [10, 1]. For each fixed alternation-depth, the model-checking problem is of polynomial complexity, but for whole of L_μ the best current algorithms still have complexity exponential in a function of the alternation depth.

It is therefore of both practical and theoretical interest to reduce, whenever possible, the number of alternations used to express a property. Even though the problem must be at least EXPTIME-hard, in practice model checking is likely to benefit from the one-time cost of reducing a formula to its simplest form, especially since the size of the formula is unlikely to dominate the runtime complexity of the model checking. However, only properties expressible in modal logic or with a single type of fixpoint operator are currently known to be recognisable. In general, for a given Φ, finding an equivalent Ψ with smallest alternation depth is one of the main open problems surrounding the modal μ calculus.

24th EACSL Annual Conference on Computer Science Logic (CSL 2015).
Editor: Stephan Kreutzer; pp. 457–471

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Here we focus on the lowest levels of the alternation hierarchy, which are known to be decidable. The class *ML* of properties expressible in modal logic, L_μ without fixpoints, resides at the base of the alternation hierarchy. These are properties which dictate some behaviour in the initial fragment of a structure, up to fixed depth. Otto [13] showed that properties of this class are recognisable via a reduction to S2S, the monadic second order logic over binary trees. Küsters and Wilke showed in [8] that the problem of deciding whether a property of L_μ can be expressed with only least fixpoints, or, by duality, only greatest fixpoints, is EXPTIME-complete. Their proof first constructs a bottom-up tree automaton of which the states correspond to sets of subformulas based on the L_μ formula. Roughly speaking, the bottom-up automaton accepts a structure if it has an initial fragment such that every completion admits a valid assignment of automaton states to its nodes. This automaton is closed under bisimulation if and only if it is Σ_1^μ definable and equivalent to the original formula.

Both Otto's, and Küsters and Wilke's results focus on deciding whether a property is expressible with a formula of the lower class, but they pay little heed to the target formula. The Σ_1^μ-formula is described in the technical report [9] as part of the proof of decidability of the first alternation level. Unfortunately the transformation can incur a double-exponential blow-up in the size of the formula. From a model-checking point of view, this is problematic: not only is the transformation into a Σ_1^μ-formula non trivial, but the transformation does not reduce the complexity of the model-checking procedure. The formula is also quite complex and does not necessarily resemble the original formula, so it is difficult to follow how the redundant fixpoints were eliminated. Otto does not describe the target *ML* formula at all but it seems that if one can be extracted from the decision procedure for *ML*, it will also be based on a power-set construction around subformulas of the original formula.

This paper puts the focus on the relation between a formula and its equivalent formulas in lower alternation classes. It describes $\Psi^{ML}, \Psi^{\Pi_1^\mu}$ and $\Psi^{\Sigma_1^\mu}$, formulas based on, and syntactically close to Ψ such that Ψ is semantically equivalent to Ψ^C if Ψ is semantically in the class C. We show that the required transformations into a *ML*, Π_1^μ or Σ_1^μ formula are conceptually very simple and easily implementable. The formula Ψ^{ML} is perhaps as one could anticipate: if Ψ is semantically in *ML*, then there is some m such that Ψ is equivalent to the formula obtained by approximating all fixpoints to their m^{th} stage and truncating the resulting formula at modal depth m. As it turns out, m needs to be at most exponential in the length of the formula. Interestingly, the potential blow-up in the size of the formula is not accidental: there are properties which are semantically modal but can be expressed with much shorter Σ_1^μ-formulas than *ML*-formulas. We show that in this sense, Σ_1^μ is at least sub-exponentially more concise than *ML*. There is a clear trade-off between syntactic complexity and formula length. From the model checking point of view, this means that if a formula has high modal depth, it may be wise to retain some fixpoint operators which will keep the size of the formula down. In contrast to Ψ^{ML}, the most interesting aspect of $\Psi^{\Pi_1^\mu}$ is perhaps its simplicity. As long as Ψ is given in disjunctive form, $\Psi^{\Pi_1^\mu}$ and its dual $\Psi^{\Sigma_1^\mu}$ are at most as large as Ψ: for disjunctive formulas, Π_1^μ and Σ_1^μ are perfectly concise in the sense that using further alternations to express the same property does not reduce the size of the formula. This is significant in that the transformation from Ψ to $\Psi^{\Pi_1^\mu}$ results in a genuinely simpler formula instead of a formula in which alternations are eliminated at the cost of conciseness. The exponential complexity of the resulting decision procedure which compares Ψ with $\Psi^{\Pi_1^\mu}$ is also optimal. The transformation itself is also noteworthy: it consists roughly speaking of replacing every μ-operator with either $\bot$ or a ν-operator. In other words, in L_μ, any satisfiable μ-subformula is either necessary or interchangeable with the identical ν-subformula.

The key to the transformation into $\Psi^{\Pi^{\mu}_{1}}$ is the use of disjunctive form. Disjunctive form, introduced in [4], is a syntactic constraint on conjunctions and universal modalities. It has been used in the context of tableau methods to decide satisfiability for example but as this paper shows, it is also a promising tool for syntactic manipulations.

2 Preliminaries

▶ **Definition 1** (Modal μ)**.** Given a set of atomic propositions $Prop = \{P, Q, ...\}$ and a set of fixpoint variables $Var = \{X, Y, ...\}$, the syntax of L_μ is given by:

$$\phi := P \mid X \mid \neg P \mid \phi \wedge \phi \mid \phi \vee \phi \mid \Diamond\phi \mid \Box\phi \mid \mu X.\phi \mid \nu X.\phi \mid \bot \mid \top$$

This definition only allows formulas in positive from: negation is only applied to propositional variables. Positivity does not restrict the expressivity of the logic. A formula is guarded if every fixpoint variable is within its binding in the scope of a modality. As is well documented in the literature [11, 7] every L_μ formula is equivalent to a formula in guarded form. Without loss of expressivity, we therefore restrict ourselves to L_μ in guarded positive form. For the sake of clarity, we only consider the uni-modal L_μ but expect the multi-modal case, as defined in [2] for example, to behave broadly speaking similarly.

Notation

If $\phi(X)$ is a formula, we write $\phi(\psi)$ for the formula ϕ where every occurrence of the variable X is replaced with ψ. For readability, if ϕ is the binding formula of the fixpoint variable X as in $\mu X.\phi$, then $\phi(\psi)$ is also ϕ with X substituted by ψ.

Formulas of L_μ are evaluated on transition systems, referred to as structures, represented by potentially infinite trees annotated with propositions.

▶ **Definition 2** (Structures)**.** A structure $\mathcal{M} = (S, s_0, R, P)$ consists of a set of states S, rooted at some initial state $s_0 \in S$, and a successor relation $R \subseteq S \times S$ between the states. Every state s is associated with a set of propositions $P(s) \subseteq Prop$ which it is said to satisfy. In this document it is sufficient for us to consider finitely branching structures, so we require that nodes only have finitely many successors. It is well-known that any structure can be represented as a potentially infinite tree. To ease the manipulation of structures, we adopt this representation.

For clarity and conciseness, we give the semantics directly in terms of parity games – the equivalence between these and the usual semantics is a standard result. For a presentation of the standard semantics of L_μ and a proof of the equivalence to the above, see for example Bradfield and Stirling [2].

▶ **Definition 3** (Parity games)**.** A parity game is a potentially infinite two-player game on a graph $\mathcal{G} = (V_0, V_1, E, v_I, \Omega)$ of which the vertices consist of two disjoint sets, V_0 and V_1 belonging to the players Even and Odd respectively, and are annotated with positive integer priorities bounded by some maximal priority q, via $\Omega : V_0 \cup V_1 \to \{0, 1, ..., q\}$. Player Even and her opponent, player Odd, move a token along the edges $E \in (V_0 \cup V_1) \times (V_0 \cup V_1)$ of the graph starting from an initial position $v_I \in V_0 \cup V_1$, each choosing the next position when the token is on a vertex in their partition. Some positions p might have no successors in which case they are winning for the player of the parity of $\Omega(p)$. A play consists of the potentially infinite sequence of vertices visited by the token. For finite plays, the last visited

parity decides the winner of the play. For infinite plays, the parity of the lowest priority visited infinitely often decides the winner of the game: Even wins if the lowest priority visited infinitely often is even; otherwise Odd wins. Note that in the literature, the highest priority is sometimes used, equivalently, as the most significant priority.

The winner of a parity game is defined in terms of existence of winning strategies. Strategies in general can depend on the history of the game, but in the case of parity games positional strategies which depend on the current position alone are sufficient, so we define strategies as mappings from position to position.

▶ **Definition 4** (Positional Strategies)**.** A positional strategy σ for one of the players in a parity game $\mathcal{G}$ is a mapping from the Player's positions V_0 or V_1 in the game to a valid successor position. A play respects a Player's strategy σ if the successor positions in the play belonging to the Player are those dictated by σ. If σ is Even's strategy and τ is Odd's strategy, then there is a unique play $\sigma \times \tau$ respecting both strategies. The winner of the parity game at a position is the player who has a strategy σ, said to be a winning strategy, such that they win $\sigma \times \tau$ from that position for any counter-strategy τ. A strategy σ is said to reach a position if there is a counter-strategy τ such that the position is along the play $\sigma \times \tau$.

Parity games are positionally determined: for every position either Even or Odd has a winning positional strategy [3]. This means that strategies gain nothing from looking at the whole play rather than just the current position. As a consequence, we may take a strategy to be memoryless: it maps each position of a player to a successor.

For any modal μ formula ϕ and a structure $\mathcal{M}$ we define a parity game $\mathcal{M} \times \phi$, constructed in polynomial time, and say that $\mathcal{M}$ satisfies ϕ, written $\mathcal{M} \models \phi$, if and only if Even has a winning strategy in $\mathcal{M} \times \phi$.

▶ **Definition 5** (Model-checking parity game)**.** For any formula ϕ of L_μ, taken to be in positive form, and a model $\mathcal{M}$, define a parity game $\mathcal{M} \times \phi$ with positions (s, ψ) where s is a state of $\mathcal{M}$ and ψ is a subformula of ϕ. The initial position is (s_0, ϕ) where s_0 is the root of $\mathcal{M}$. Positions (s, ψ) where ψ is a disjunction or a formula starting with an existential modality $\Diamond$ belong to Even while conjunctions and formulas starting with a universal modality $\Box$ belong to Odd. Other positions have at most one successor so their owner is irrelevant; let them be Even's. There are edges from $(s, \psi \vee \psi')$ and $(s, \psi \wedge \psi')$ to both (s, ψ) and (s, ψ'); from $(s, \mu X.\phi)$ and $(s, \nu X.\phi)$ to (s, ϕ); from (s, X) to $(s, \nu X.\psi)$ if X is bound by ν, or $(s, \mu X.\psi)$ if it is bound by μ; finally, from $(s, \Diamond\psi)$ and $(s, \Box\psi)$ to every (s', ψ) where (s, s') is an edge in the model $\mathcal{M}$. Positions (s, P), $(s, \neg P)$, $(s, \top)$ and $(s, \bot)$ have no successors. The parity function assigns an even priority to $(s, \top)$ and also to (s, P) if P satisfies s in $\mathcal{M}$ and to $(s, \neg P)$ if s does not satisfy P in $\mathcal{M}$; otherwise (s, P) and $(s, \neg P)$ receive odd priorities, along with $(s, \bot)$. Fixpoint variables are given distinct priorities such that ν-bound variables receive even priorities while μ-bound variables receive odd priorities. Furthermore, whenever X has priority i, Y has priority j and $i > j$, X must not appear free in the formula ψ binding Y in $\mu Y.\psi$ or $\nu Y.\psi$. In other words, inner fixpoints receive higher, less significant priorities while outer fixpoint receive low priorities. Other nodes receive a priority *max* which is larger than any of the priorities assigned to fixpoint nodes. This ensures that these will never be the lowest priority seen infinitely often.

We now use parity games to define the semantics of L_μ.

▶ **Definition 6** (Satisfaction relation)**.** A structure $\mathcal{M}$, rooted at s_0 is said to satisfy a formula Ψ of L_μ, written $\mathcal{M} \models \Psi$ if and only if the Even player has a winning strategy from (s_0, Ψ)

in $\mathcal{M} \times \Psi$. $\mathcal{M}$ satisfies a subformula ϕ of Ψ if it satisfies the formula ϕ where free fixpoint variables X are recursively replaced with their fixpoint binding $\mu X.\phi$ or $\nu X.\phi$ from Ψ. This is the case if and only if Even has a winning strategy from (s, ϕ) in $\mathcal{M} \times \Psi$.

Formulas are semantically equivalent if they are satisfied by exactly the same structures.

▶ **Definition 7** (Modal Logic, Π_1^μ and Σ_1^μ)**.** *ML* is the class of properties of structures expressible in modal logic, that is to say in L_μ without any fixpoint operators. A formula without fixpoint operators is said to be modal and has a modal depth which is the greatest number of nested modal operators in it. Π_1^μ is the class of properties expressible by a formula in positive form without using the least fixpoint operator μ and Σ_1^μ is the class of properties expressible by a formula in positive form without using the greatest fixpoint operator ν. If a formula does not contain μ, ν or both it is said to be *syntactically* in Π_1^μ, Σ_1^μ or *ML* respectively.

Beyond *ML*, Π_1^μ and Σ_1^μ, the syntactic complexity of formulas is measured by the number of alternations between least and greatest fixpoint operators. This is the alternation depth of a formula and corresponds to the number of priorities needed in the model-checking parity game. A precise definition of the alternation depth is given for example in [2]. The fragments of L_μ with bounded alternation depth form the alternation hierarchy, which is known to be strict: for each level, there are formulas which cannot be expressed with a formula of lower alternation depth [1, 10].

If a formula is equivalent to a formula syntactically in some alternation level, it is said to be semantically in that alternation level. Thus a formula of high syntactic alternation level may be of low semantic alternation level.

Our concern is to decide whether a formula is semantically in one of Π_1^μ, Σ_1^μ or *ML* and produce an equivalent formula syntactically in the appropriate alternation level.

3 The formula for *ML*

The modal fragment of L_μ, or *ML*, was shown by Otto to be decidable: for any formula of L_μ, we can decide whether there is an equivalent modal formula [13]. This section proposes a proof of decidability by formula construction: given a guarded formula Ψ, it presents a formula Ψ^{ML} in *ML* which Ψ is equivalent to if and only if Ψ is semantically a modal formula. The crux of the argument is that a semantically modal formula Ψ can only reach depth $2^{2|\Psi|}$ in any structure and therefore, if a formula is equivalent to a modal formula, it is sufficient to first approximate all fixpoints to the $2^{2|\Psi|}$-th stage of induction and then truncate the formula at modal depth $2^{2|\Psi|}$. At first sight this might seem like a wasteful solution since the size of the formula increases as it is unfolded. However, Example 20 shows that there are formulas which cannot be expressed in modal logic without at least a sub-exponential increase in formula size. This proves that approximating the fixponts to their $2^{2|\Psi|th}$ stage of induction is hardly excessive.

The first two definitions fix the notation for measuring the depth of a structure and approximating fixpoints. Note that all structures are represented as trees since we are interested in how far into a structure a formula can reach and this is easier to do when reasoning about trees rather than graphs.

▶ **Definition 8** (Rank and depth)**.** The rank of a state without successors is 0. The rank of a state with finitely many successors is $h + 1$ where h is the maximal rank amongst the state's successors. The depth of the root of a structure is 0; otherwise the depth of a state is one greater than the depth of its parent.

The rank of a finite tree is the rank of its root and corresponds to the length of the longest path in the tree.

The next definition formalises the notion of simultaneously evaluating all fixpoints to their n^{th} stage of induction.

▶ **Definition 9** (Approximants). Let $\mu X^0.\phi = \bot$ and $\mu X^n.\phi(X) = \phi(\mu X^{n-1}.\phi(X))$; let $\nu X^0.\phi = \top$ and $\nu X^n.\phi(X) = \phi(\nu X^{n-1}.\phi(X))$.
Then, using the notation $\phi[a/b]$ to mean ϕ where all instances of b are substituted with a, and *fixpoints*(ϕ) is the set of fixpoint variables in ϕ, let $\phi_n = \phi[\nu X^n/\nu X; \mu X^n/\mu X]_{\forall X \in \mathit{fixpoints}(\phi)}$, the formula ϕ where every fixpoint μX or νX is substituted with its n^{th} approximation μX^n or νX^n. See, e.g., [2].

We can then show easily enough that for trees of bounded height n, there is never any need to go beyond the n^{th} stage of induction. The intuition is as follows: without change in semantics, we can unfold all the fixpoints in ϕ all of n times each to obtain a formula equivalent to ϕ which only differs syntactically from ϕ_n at modal depth greater than n. On trees of bounded height n, the model-checking parity game will never reach this point for either formula. Therefore, both games for ϕ and ϕ_n must agree.

▶ **Lemma 10.** *If ϕ is guarded, ϕ and ϕ_n agree on trees of rank bounded by n.*

Proof. The game for ϕ_n is similar to the game for ϕ except for the additional rule that each priority p has a counter attached to it which counts how many times p occurs in the play without a smaller priority occurring in between. If a counter for an odd priority reaches $n+1$, Even loses immediately; if an even priority counter reaches $n+1$, then Odd loses immediately. If $\mathcal{M}$ is a tree of bounded rank, that is to say with a longest path of length no more than n, and ϕ is guarded, all plays in $\mathcal{M} \times \phi$ visit at most n states and do not visit a position (s, ψ) more than once. Therefore no priority is seen more than n times: no counter can reach $n+1$, so Even wins $\phi_n \times \mathcal{M}$ if and only if she can win $\phi \times \mathcal{M}$. Hence, on $\mathcal{M}$ a tree of rank at most n, ϕ and ϕ_n must agree. Write $\mathcal{M} \models \phi$ if and only if $\mathcal{M} \models \phi_n$. ◀

The formula ϕ_n is a modal formula but it may have modal depth greater than n, for example if a fixpoint is guarded by more than one modality or if it has interacting fixpoints. We will therefore define a truncating operation which reduces the modal depth of a formula to n.

▶ **Definition 11.** Let the formula ϕ^n be the one obtained from a modal formula ϕ by replacing subformulas $\Box\psi$ of modal depth n or larger with $\top$ and $\Diamond\psi$ of modal depth n or larger with $\bot$.

▶ **Lemma 12.** *Let Φ be guarded. Then $\mathcal{M} \models \phi^n$ iff $\mathcal{M}^n \models \phi$ where $\mathcal{M}^n$ is the infinite tree of $\mathcal{M}$ truncated at depth n. That is to say, ϕ^n is true in $\mathcal{M}$ iff the initial three of height n of $\mathcal{M}$ satisfies ϕ.*

Proof. First note that in the model checking parity game of modal formulas, a state at depth n can only be reached at a subformula that is itself at modal depth n. Assuming $\mathcal{M} \models \phi^n$, Even has a winning strategy σ in $\mathcal{M} \times \phi^n$ to prove it. This game is identical to $\mathcal{M} \times \phi$ until a position s at depth n is reached at $\bot$ or $\top$ instead of $\Diamond\psi$ or $\Box\psi$ respectively. If Even can win, her strategy cannot reach any position $(s, \bot)$. The game $\mathcal{M}^n \times \phi$ is also identical to $\mathcal{M} \times \phi$ until a position at depth n is reached. The strategy σ is winning in $\mathcal{M}^n \times \phi$ since it can avoid $(s, \Diamond\psi)$ positions where s it at depth n and positions $(s, \Box\psi)$ are automatically winning for states s at depth n since they have no successors in $\mathcal{M}^n$.
Conversely, assume Even has a winning strategy σ in $\mathcal{M}^n \times \phi$. She can use this strategy

in $\mathcal{M} \models \phi^n$ until it reaches positions (s, ψ) where s is at depth n. Since these are leaves, her winning strategy does not reach any state $(s, \Diamond\psi)$ where s is at depth n. In $\mathcal{M} \models \phi^n$ her strategy σ therefore only reaches final positions (s, P) and $(s, \top)$ where s is at depth n, which are winning for her. A strategy is therefore winning in $\mathcal{M} \times \phi^n$ if and only if it is winning in $\mathcal{M}^n \times \phi$ and therefore $\mathcal{M} \models \phi^n$ iff $\mathcal{M}^n \models \phi$. ◀

▶ **Example 13.** Consider the modal formula $\phi = A_0 \wedge \Diamond A_1 \wedge \Box(\Diamond\Diamond A_2 \vee \Box A_3) \wedge \Box\Box\Box A_4$. Then $\phi^3 = A_0 \wedge \Diamond A_1 \wedge \Box(\Diamond\bot \vee \Box A_3) \wedge \Box\Box\top$ is true in $\mathcal{M}$ iff its initial tree of height 3 satisfies ϕ. Similarly $\phi^2 = A_0 \wedge \Diamond A_1 \wedge \Box(\bot \vee \top) \wedge \Box\top$ is true in $\mathcal{M}$ if its initial tree of height 2 satisfies ϕ. Finally $\phi^1 = A_0 \wedge \bot \wedge \top = \bot$, which is reasonable since the root of $\mathcal{M}$ cannot satisfy $\Diamond A_1$ without having any successors.

We use the following lemma, which Otto also uses in [13]. The intuition is that if a formula is equivalent to a *ML*-formula of modal depth m, then what happens beyond depth m in a structure can have no effect on whether the formula holds in this structure or not. The semantic modal depth of a semantically *ML* formula is the least modal depth of any equivalent *ML* formula. Note that the syntactic alternation-depth of a formula is irrelevant to its semantic modal depth but only semantically *ML* formulas have a finite modal depth.

▶ **Lemma 14.** *If ϕ is guarded and of semantic modal depth m, then $\mathcal{M} \models \phi$ iff $\mathcal{M}^m \models \phi$ where $\mathcal{M}^m$ is the infinite tree of $\mathcal{M}$ truncated at depth m.*

Proof. If ϕ is guarded and of semantic modal depth m, there are formulas ψ equivalent to ϕ of syntactic modal depth m. The model checking parity game for a modal formula has no infinite paths in it. Furthermore for a formula of modal depth m, a play can visit at most m distinct states. As a result, in the games $\mathcal{M} \times \psi$, only positions containing states no deeper than m are reachable: $\mathcal{M} \times \psi$ and $\mathcal{M}^m \times \psi$ are identical. Since ψ is equivalent to ϕ, $\mathcal{M} \models \phi$ iff $\mathcal{M}^m \models \phi$. ◀

We can now show that if ϕ is of semantic modal depth m, then it is equivalent to the formula ϕ^m_m where fixpoints are first approximated to the m^{th} stage of induction as detailed in Definition 9 and then truncated at modal depth m as per Definition 11.

▶ **Theorem 15.** *If ϕ is guarded and of semantic modal depth m, then ϕ is equivalent to ϕ^m_m.*

Proof. Let ϕ be a guarded L_μ formula equivalent to a modal formula of modal depth m. Then $\mathcal{M} \models \phi$ iff $\mathcal{M}^m \models \phi$. However, ϕ agrees with ϕ_m on all trees of height at most m. Therefore the following are equivalent:
(1) $\mathcal{M} \models \phi$
(2) $\mathcal{M}^m \models \phi$
(3) $\mathcal{M}^m \models \phi_m$
(4) $\mathcal{M} \models \phi^m_m$
The conditions (1) and (2) are equivalent since ϕ is semantically modal of depth m, as per Lemma 14. Then (2) and (3) are equivalent since ϕ and ϕ_m have the same truth-value on $\mathcal{M}^m$, from Lemma 10. Finally, (3) and (4) are equivalent by definition of ϕ^m_m. ◀

Next we aim to show that m can be calculated from ϕ, using an argument similar to the one used by Otto [13]. The argument relies on labelling the states of structures with the subformulas of ϕ it satisfies and noting that the successors of a state can freely change as long as the set of successor-labels remains the same without affecting the formulas the state satisfies. The crux of the argument is that if two structures only differ at very high depth, but one satisfies ϕ and the other one does not, then the state labels must repeat themselves

before the point at which the structures differ. Then we can duplicate a portion of the branch leading to the difference in order to create structures which are differentiated even deeper but still only one of them satisfies ϕ. This shows that if ϕ is modal, its modal depth cannot be deeper that the point at which the state labels need to start repeating themselves. $2^{2|\phi|}+1$ is an upper bound for that point.

The next lemma uses the fact that in an infinite tree, any subtree rooted at s can be replaced with a distinct subtree rooted at s' without affecting the subformulas of Ψ satisfied above depth s as long as the subtrees rooted at s and s' agree on all subformulas of Ψ. For a proof, see for example [6]. This should be clear from the notion that whether a state satisfies a subformula of Φ depends only on the propositional variables that state satisfies and the subformulas satisfied by its successor states.

▶ **Definition 16.** Let $\mathcal{M} = (M, i_M, E_M, P_M)$ be an infinite tree and t be a state of $\mathcal{M}$, and let $\Psi \in L_\mu$. We denote by $\alpha^\Psi_M(t)$ the set of subformulas of Ψ satisfied by the state t in $\mathcal{M}$.

▶ **Lemma 17** (Consistent labelling). *Let there be two disjoint trees, $\mathcal{M} = (M, i_M, E_M, P_M)$ and $\mathcal{M}' = (M', i_{M'}, E_{M'}, P_{M'})$, and a sentence $\Psi \in L_\mu$. Let s and s' be states of $\mathcal{M}$ and $\mathcal{M}'$ respectively, such that $\alpha^\Psi_M(s) = \alpha^\Psi_{M'}(s')$, and let v be the predecessor of s in M.*

Replace the edge e from v to s within $\mathcal{M}$ by a new edge e' from v to s' to obtain a new model $\mathcal{N}$ built from parts of $\mathcal{M}$ and $\mathcal{M}'$. More precisely, $\mathcal{N} = (N, i_N, E_N, P_N)$ with

- *$N = M \setminus \{u \in M \mid u \text{ extends or is equal to } s\} \cup \{u \in M' \mid u \text{ extends or is equal to } s'\}$ as set of states;*
- *$i_N = i_M$ as initial node;*
- *$E_N = (N \times N \restriction E_M) \cup (N \times N \restriction E_{M'}) \cup \{e'\}$ as set of edges, where $\restriction$ denotes restriction; and*
- *$P_N = (P_M \restriction N \cup P_{M'}) \restriction N$ as propositional variables.*

Then, since $N \subseteq M \uplus M'$ (where $\uplus$ denotes disjoint union), the labelling $(\alpha^\Psi_M \cup \alpha^\Psi_{M'}) \restriction N$ is defined on all states of $\mathcal{N}$ as well. Moreover, for all $s \in N$ we have $(\mathcal{N}, s) \models \phi$ if and only if $\phi \in ((\alpha^\Psi_M \cup \alpha^\Psi_{M'}) \restriction N)(s)$, meaning that $(\alpha^\Psi_M \cup \alpha^\Psi_{M'}) \restriction N$ is identical to α^Ψ_N.

▶ **Lemma 18.** *Let ϕ be guarded and semantically modal, i.e. ϕ is equivalent to a formula in ML. Then the semantic modal depth m of ϕ is bounded above by $2^{2|\phi|}+1$.*

Proof. Assume $m > 2^{2|\phi|}+1$ to be the semantic modal depth of ϕ. Then there exists a tree $\mathcal{M}$ of height $2^{2|\phi|}+1$ which is the prefix of two models $\mathcal{M}_1$ and $\mathcal{M}_2$ such that $\mathcal{M}_1 \models \phi$ and $\mathcal{M}_2 \not\models \phi$. That is to say, for every state s of $\mathcal{M}$, there are states s_1 and s_2 in $\mathcal{M}_1$ and $\mathcal{M}_2$ respectively such that s, s_1 and s_2 agree on propositions and for all inner nodes of $\mathcal{M}$, s' is a successor of s if and only if s'_1 is a successor of s_1, if and only if s'_2 is a successor of s_2. If d is maximal such that $\mathcal{M}_1$ and $\mathcal{M}_2$ agree up to depth d, write $agree(\mathcal{M}_1, \mathcal{M}_2) = d$. To start with, $agree(\mathcal{M}_1, \mathcal{M}_2) > 2^{2|\phi|}$ since $\mathcal{M}_1$ and $\mathcal{M}_2$ agree on their prefix $\mathcal{M}$ of rank $2^{2|\phi|}+1$.

Label every state s of $\mathcal{M}$ with a set $\alpha^\phi_{M_1}(s)$ consisting of subformulas of ϕ which are true in $\mathcal{M}_1$ and a set $\alpha^\phi_{M_2}(s)$ consisting of subformulas of ϕ which are true in $\mathcal{M}_2$. For each branch of $\mathcal{M}$, that is to say a path from the root of $\mathcal{M}$, if the branch is longer than $2^{2|\phi|}$, there are two states a, b in $\mathcal{M}$ along the branch such that $\alpha^\phi_{M_1}(a) = \alpha^\phi_{M_1}(b)$ and $\alpha^\phi_{M_2}(a) = \alpha^\phi_{M_2}(b)$. For each branch i, choose b^i to be the first state on a branch which has an ancestor a^i such that $\alpha^\phi_{M_1}(a^i) = \alpha^\phi_{M_1}(b^i)$ and $\alpha^\phi_{M_2}(a^i) = \alpha^\phi_{M_2}(b^i)$. Note that for any pair of branches i and j, either $b^i = b^j$ or b^i and b^j are not reachable from one another.

For each branch i and its states a^i and b^i, let $a^{i\prime}_1$ be the root of a distinct copy of the subtree in $\mathcal{M}_1$ rooted at a^i. Similarly, let $a^{i\prime}_2$ be the root of a distinct copy of the subtree rooted at a^i in $\mathcal{M}_2$. Let $\mathcal{M}'_1$ be obtained from $\mathcal{M}_1$ where for each branch i, the state b^i is

replaced with $a^{i\prime}_1$ and its induced subtree; let $\mathcal{M}_2^{'}$ be obtained from $\mathcal{M}_2$ where b^i is replaced with $a^{i\prime}_2$ and its induced subtree. Note that these transformations do not affect each other: recall that each b^i is on a distinct branch and is replaced with a subtree of the original structure. Since $\alpha^{\phi}_{M_1}(a^{i\prime}_1) = \alpha^{\phi}_{M_1}(b^i)$ and $\alpha^{\phi}_{M_2}(a^{i\prime}_2) = \alpha^{\phi}_{M_2}(b^i)$, all states preserve their labels and we know that $\mathcal{M}'_1 \models \phi$ and $\mathcal{M}'_2 \not\models \phi$, from Lemma 17.

We now show that if $\mathcal{M}_1$ and $\mathcal{M}_2$ agree up to depth d, then $\mathcal{M}'_1$ and $\mathcal{M}'_2$ agree up to depth $d+1$. Let i be a branch in $\mathcal{M}$ of length d such that i is extended differently in models $\mathcal{M}_1$ and $\mathcal{M}_2$. Since $depth(b^i) > depth(a^i)$ the models $\mathcal{M}'_1$ and $\mathcal{M}'_2$ agree along all extensions of i to depth $d - depth(a^i) + depth(b^i) > d$. That is to say $\mathcal{M}'_1$ and $\mathcal{M}'_2$ agree at least up to $d+1$. This establishes $agree(\mathcal{M}'_1, \mathcal{M}'_2) > agree(\mathcal{M}_1, \mathcal{M}_2)$. In $z = m - d$ many steps, we will reach models $\mathcal{M}^z_1$ and $\mathcal{M}^z_2$ such that $agree(\mathcal{M}^z_1, \mathcal{M}^z_2) \geq m$ but $\mathcal{M}^z_1 \models \phi$ and $\mathcal{M}^z_2 \not\models \phi$. This contradicts m being the modal depth of ϕ. ◀

▶ **Corollary 19.** *Whether a guarded formula ϕ is equivalent to a modal formula can be decided by testing whether ϕ is equivalent to $\phi^{2^{|\phi|}+1}_{2^{|\phi|}+1}$.*

Proof. From the previous lemma, if a formula ϕ is modal, its semantic modal depth m is no greater than $2^{2|\phi|} + 1$. If $\phi \neq \phi^{2^{|\phi|}+1}_{2^{|\phi|}+1}$, then ϕ must disagree with $\phi^{2^{|\phi|}+1}_{2^{|\phi|}+1}$ on some structure $\mathcal{M}$. However, the two model checking games $\mathcal{M} \times \phi$ and $\mathcal{M} \times \phi^{2^{|\phi|}+1}_{2^{|\phi|}+1}$ are identical on plays which do not reach states deeper than $2^{|\phi|} + 1$, which, since $m \leq 2^{|\phi|} + 1$, contradicts the fact that the modal depth of ϕ is m. ◀

The most surprising aspect of this result is perhaps the exponential modal depth. This is not due to the authors' laziness: formulas with fixpoints can indeed be at least sub-exponentially more compact than the equivalent modal formulas. The following exhibits syntactically Σ^{μ}_1-formulas but semantically modal formulas with sub-exponential modal depth. The idea of these formulas is to require a series of propositional variables to occur at different frequencies until they all occur at the same time. The modal depth of the formula is then the least common multiple of the frequencies.

▶ **Example 20.** There is a family of formulas $\Phi_n \in \Sigma^{\mu}_1$ which are semantically modal but have modal depth $\Omega(2^{\sqrt{n}})$ in the length of Φ_n.

Proof. Write $\Box^n$ for $\Box ... \Box$ repeated n times. The formula $\mu X.(A \wedge (\Box^n X \vee B)$ states that A occurs every n^{th} state on any path until B also occurs at a state whose depth is a multiple of n. By combining such formulas we can write $[\Box^a \mu X.A \wedge (\Box^a X \vee (B \wedge C))] \wedge [\Box^b \mu X.B \wedge (\Box^b X \vee (A \wedge C))] \wedge [\Box^c \mu X.C \wedge (\Box^c X \vee (A \wedge B))]$ which sets the frequencies at which A, B and C are seen until they are seen simultaneously. This formula is modal since if it is true, at the latest at depth $a \times b \times c$, all of A, B and C are seen simultaneously. More precisely, its modal depth is the least common multiple of a, b and c. Generalising this, for a fixed n, let $\psi_d = \mu X.\Box^d(P_d \wedge X) \vee (\bigwedge_{i \leq n} P_i)$ be the formula stating that the proposition P_d occurs at frequency d until all propositions P_i for $i \leq n$ occur at the same time, at a depth multiple of d. Now, let $\Phi_n = \bigwedge_{d \leq n} \psi_d$. The modal depth of Φ_n is the least common multiple of the integers up to n, written $lcm(n)$. For sufficiently large n, $lcm(n) > 2^n$ [12] so the formula Φ_n is of length $O(n^2)$ and has modal depth $\Omega(2^n)$ which proves the correctness of the example. ◀

4 The formulas for Π^{μ}_1 and Σ^{μ}_1

The previous section addressed how to eliminate accidental complexity from semantically modal formulas. This section studies the same question for Π^{μ}_1, the class of properties

expressible without least fixpoint operators, and its dual, Σ_1^μ. Küsters and Wilke [8] showed that it is decidable whether a formula is equivalent to a Π_1^μ formula; this section constructs the desired formula, yielding an alternative decision procedure for $\Pi_1^\mu$and Σ_1^μ. We first formalise the idea that if a property is in Σ_1^μ, then some finite initial tree is always sufficient to show that a structure satisfies the property. We then introduce disjunctive form. The final subsection shows how unnecessary fixpoints can be eliminated syntactically from formulas in disjunctive form by using the fact that Σ_1^μ formulas have finite proofs.

4.1 Properties in Π_1^μ have finite counter-proofs

In this section we characterise properties in Σ_1^μ and Π_1^μ as properties with finite proofs and counter-proofs respectively. Informally, μ-formulas express finite behaviour such as reachability – proofs of such properties are finite: once the desired state is reached, the rest of the structure is irrelevant. Dually, ν-formulas express infinite behaviour and if a structure fails to display infinite behaviour, the state at which it fails must be finitely reachable.

▶ **Lemma 21.** *Let $\mathcal{M}$ be a structure with finite branching such that $\mathcal{M} \not\models \Psi$. If $\Psi \in \Pi_1^\mu$ then there is some n such that for any structure $\mathcal{M}'$, if $\mathcal{M}'$ agrees with $\mathcal{M}$ up to depth n, then $\mathcal{M}' \not\models \Psi$.*

Proof. Assume Ψ is semantically in Π_1^μ and Φ is the equivalent formula with no least fixpoints. Since $\mathcal{M} \not\models \Phi$, Even has a winning strategy in $\mathcal{M} \times \neg\Phi$. Note that $\neg\Phi$ is a formula without greatest fixpoints. That means that Even has a strategy σ winning in $\mathcal{M} \times \neg\Phi$ which only agrees with finite plays. Let n be the depth of the furthest state in $\mathcal{M}$ which σ reaches – since $\mathcal{M}$ has finite branching, there is such an n. Note that agreement between $\mathcal{M}$ and $\mathcal{M}'$ up to $n+1$ requires any leaves at depth less than n in $\mathcal{M}$ to remain leaves in $\mathcal{M}'$. Now for any $\mathcal{M}'$ which agrees with $\mathcal{M}$ up to $n+1$, the strategy σ is still winning for Even so $\mathcal{M} \not\models \Psi$. ◀

4.2 Disjunctive form

Disjunctive form was introduced in [4] as a syntactic restriction to universal branching. It had been used for example to show the completeness of Kozen's axiomatisation [14]. Here we show that disjunctive forms are also a tool for simplifying syntactic manipulations. Informally, the idea of disjunctive form is to push conjunctions into the leaves and allow player Odd to make exactly one choice per state.

▶ **Definition 22.** *(Disjunctive formulas)* The set of disjunctive form formulas of (unimodal) L_μ is the smallest set $\mathcal{F}$ satisfying:

- Propositional variables and their negations, fixpoint variables and $\top$ and $\bot$ are in $\mathcal{F}$;
- If $\psi \in \mathcal{F}$ and $\phi \in \mathcal{F}$, then $\psi \vee \phi \in \mathcal{F}$;
- If $\mathcal{A}$ is a set of literals and $\mathcal{B} \subseteq \mathcal{F}$ ($\mathcal{B}$ is finite), then $\bigwedge \mathcal{A} \wedge \to\mathcal{B}$ where $\to\mathcal{B}$ is short for $(\bigwedge_{\psi\in\mathcal{B}} \Diamond\psi) \wedge \Box \bigvee_{\psi\in\mathcal{B}} \psi$ – that is to say, every formula in $\mathcal{B}$ is realised by at least one successor and every successor realises at least one of the formulas in $\mathcal{B}$;
- $\mu X.\psi$ and $\nu X.\psi$ are in $\mathcal{F}$ as long as $\psi \in \mathcal{F}$ and X only appears positively and never in a conjunction $X \wedge \alpha$ where α is another formula.

The last constraint ascertains that if $\mu X.\phi(X)$ is in disjunctive form, then $\phi(\mu X.\phi(X))$ is also in disjunctive form.

Every formula is known to be equivalent to an effectively computable formula in disjunctive form [14]. The transformation preserves guardedness.

We can now prove our key lemma about formulas in disjunctive form which exploits the restriction imposed on player Odd's choices. With an arbitrary L_μ formula, once Even has fixed a strategy, Odd may be able to choose to play to a state s at various different formulas. For example, from $(s, \Box\psi \wedge \Box\phi)$ Odd can choose to play any successor of s at either ϕ or ψ. However, with some minor assumption about the structure, once a formula is in disjunctive form and Even has fixed her strategy, Odd is much more restricted in his choices: if he chooses to play to a state s, he can only choose to play a formula fixed by Even's strategy or a literal.

First we define well behaved strategies and models in which whenever a state s is required to satisfy a modal formula $\to\mathcal{B}$, s has a distinct successor for each formula in $\mathcal{B}$. Such a well-behaved model can easily be derived from any model by duplicating the successor states of s as necessary. A well-behaved model will allow the even player to use a well-behaved strategy which chooses a distinct successor for each of the formulas Odd can choose at $\to\mathcal{B}$.

▶ **Definition 23** (Well behaved models)**.** A model $\mathcal{M}$ of Ψ is well behaved with respect to an Even's winning strategy σ in $\mathcal{M} \times \Psi$ if for each position $(s, \to\mathcal{B})$ reachable with σ, s has distinct successors s_ϕ such that $s_\phi \models \phi$ for each $\phi \in \mathcal{B}$ and σ plays s_ϕ if Odd picks ϕ from s and ϕ if Odd picks s_ϕ. Note that s may have more than one successor satisfying ϕ but σ chooses only one such successor, s_ϕ to play to whenever Odd chooses to play ϕ. A model is well-behaved if it is well-behaved with respect to some winning strategy.

Every model $\mathcal{M}$ of Ψ is bisimilar to a well behaved model of Ψ obtained by duplicating successor states as necessary. A strategy is said to be well-behaved if the model is well-behaved with respect to that strategy. Next we define the tree of Odd's playable positions induced by Even's strategy. This tree consists of the choices which Odd is left with once Even has fixed her strategy.

▶ **Definition 24** (Odd's position tree)**.** If σ is a strategy for Even, we consider the tree made out of positions belonging to Odd which are reachable by plays respecting σ. One step in the Odd's position tree corresponds to one move by Odd followed by as many moves dictated by σ as necessary to reach the next position belonging to Odd. Note that since σ is Even's strategy, Odd's position tree does not have any disjunctive positions any more, only conjunctions: all the non-leaf positions of this tree are of the form $(s, \bigwedge\mathcal{A} \wedge \to\mathcal{B})$ for some set of literals $\mathcal{A}$ and a set of formulas $\mathcal{B}$. The leaves are of the form (s, A) where A is a literal.

The following lemma shows how the syntactic constraints of disjunctive form simplify the strategies in the parity game. It is the key to our proof of decidability. It states that once Even has fixed her strategy, Odd can only reach a state s at a single formula $\bigwedge\mathcal{A} \wedge \to\mathcal{B}$. This will allow us to replace s with the root of any structure which satisfies the same formula, while preserving Even's winning strategy.

▶ **Lemma 25.** *If Ψ is in disjunctive form and $\mathcal{M}$ is the tree-representation of a well-behaved model with respect to a strategy σ, then each state of $\mathcal{M}$ appears in Odd's position tree for σ at most once at a non-leaf position.*

Proof. For a state s to appear twice at such a formula, (s, ϕ_0) and (s, ϕ_1) must be two positions of the tree with a last common ancestor, (t, ψ) where ψ has to be $\to\mathcal{B}$ for some $\mathcal{B}$ containing ϕ_0 and ϕ_1 . However, since σ is well behaved, if Odd chooses either $\phi \in \mathcal{B}$ or the successor t_ϕ, the game goes to (t_ϕ, ϕ) so each successor t_ϕ only appears in one successor position of (t, ψ). Furthermore, each non-indexed successor of t also appears in only one successor position of (t, ψ). This can therefore not be the last common ancestor of (s, ϕ_0) and (s, ϕ_1). ◀

4.3 The formula $\Psi^{\Pi_1^\mu}$

This section proves the main theorem on the constructive decidability of Π_1^μ: any semantically Π_1^μ formula in disjunctive form can be transformed into an equivalent syntactically Π_1^μ formula by changing every occurrence of μ into either ν or $\bot$.

To show this, we first select for each μ-subformula $\mu X.\phi$ in Ψ, a structure $\mathcal{M}$ such that whether $\mathcal{M}$ satisfies Ψ or not depends on a restricted set of states satisfying the formula $\mu X.\phi$, as shown in Lemma 26. We then show in Lemma 27 that if a μ-subformula $\mu X.\phi$ of Ψ is satisfiable but cannot be replaced with the corresponding ν-formula $\nu X.\phi$, then for any n we can build a twin structure for $\mathcal{M}$ agreeing with $\mathcal{M}$ up to n but disagreeing on Ψ. This implies that Ψ is not a Π_1^μ formula, as Lemma 21 shows that Π_1^μ formulas have finite counter-proofs. This leaves us with two scenarios: either the μ-subformula is unsatisfiable, in which case it can be replaced by $\bot$, using Lemma 28, or the μ-formula can be replaced with the corresponding ν-formula. In either case, we can turn any semantically Π_1^μ formula into a syntactically Π_1^μ formula by replacing μ-subformulas with either $\bot$ or the dual ν-formula.

Notation. Let $\Psi(\psi)$ be a formula in disjunctive form which contains a subformula ψ. We will write $\Psi(\psi')$ for the formula in which ψ is substituted with the formula ψ'. With this notation, we will use formulas related to Ψ in order to specify structures where the players' strategies must exhibit some desired behaviours. For example, if for some structure $\mathcal{M}$, Odd can win $\mathcal{M} \times \Psi(\bot)$, then Even can only win $\mathcal{M} \times \Psi(\mu X.\phi)$ by playing eventually to $\mu X.\phi$.

In the following lemma we show that for a structure to satisfy $\neg\Psi(\phi) \wedge \Psi(\top)$ means that Odd can win the game for $\Psi(\phi)$, but only by playing to a position (s, ϕ) for some s in a set S. Then, if states of S are substituted with new substructures, Odd may only win if he can win from one of the new substructures at ϕ.

▶ **Lemma 26.** *Let Ψ be a formula in disjunctive guarded form with a subformula ϕ. If $\mathcal{M}$ is a structure such that $\mathcal{M} \models \neg\Psi(\phi) \wedge \Psi(\top)$ and $\mathcal{M}$ is well-behaved for $\Psi(\top)$, then there is a non-empty set of states S in $\mathcal{M}$ such that in $\mathcal{M} \times \Psi(\phi)$ each of Odd's winning strategies reaches (s, ϕ) for some $s \in S$– that is to say, for each of Odd's winning strategies τ there is a counter strategy σ such that (s, ϕ) is on the play $\tau \times \sigma$ for some $s \in S$. Furthermore, if every state s_i of S is replaced with some state t_i, yielding a new model $\mathcal{M}'$, Odd only wins in $\mathcal{M}' \times \Psi(\phi)$ if Odd wins from (t_i, ϕ) in the same game for some t_i.*

Proof. If Even wins $\mathcal{M} \times \Psi(\top)$ but Odd wins $\mathcal{M} \times \Psi(\phi)$, then Odd cannot win $\mathcal{M} \times \Psi(\phi)$ with a strategy which avoids ϕ, otherwise the same strategy would be winning in $\mathcal{M} \times \Psi(\top)$. Let τ be one of Odd's winning strategies in $\mathcal{M} \times \Psi(\phi)$ and let σ be Even's well-behaved winning strategy in $\mathcal{M} \times \Psi(\top)$. Since $\mathcal{M} \times \Psi(\top)$ is identical to $\mathcal{M} \times \Psi(\phi)$ until a play reaches ϕ, the strategy σ is also an initial strategy in $\mathcal{M} \times \Psi(\phi)$, defined until ϕ is reached. The play $\tau \times \sigma$ must reach ϕ because otherwise it would have to be winning for Even due to it being identical to a play respecting her winning strategy in $\mathcal{M} \times \Psi(\top)$. Let s_τ be the first state at which the play $\tau \times \sigma$ reaches ϕ in $\mathcal{M} \times \Psi(\phi)$. Then $S = \{s_\tau | \tau$ is a winning strategy for Odd $\}$ is the set such that in $\mathcal{M} \times \Psi(\phi)$ each of Odd's winning strategies reaches (s, ϕ) for some $s \in S$.

For the second part of the lemma, first observe that if Even wins from (t_i, ϕ) for all i, then Odd cannot use any of his winning strategies from $\mathcal{M} \times \Psi(\phi)$ to win in $\mathcal{M}' \times \Psi(\phi)$ since if Even initially plays according to σ, the play reaches (t_i, ϕ) from where Even has a winning strategy. As a result, Odd cannot avoid all t_i without losing. From Lemma 25 we know that each t_i is only seen at position (t_i, ϕ) so not only can Odd not avoid all t_i, Odd cannot avoid all (t_i, ϕ) without losing. Hence, if Odd loses from (t_i, ϕ) for all i, Odd loses in $\mathcal{M}' \times \Psi(\phi)$. ◀

We can now prove the main result: to obtain the syntactically Π_1^μ formula equivalent to a semantically Π_1^μ formula in guarded disjunctive form, it is sufficient to replace each least fixpoint with either $\bot$ or a greatest fixpoint. The crux is to show that each μ-binding in a semantically Π_1^μ formula can either be replaced by $\bot$ or ν. The following lemma identifies two cases. The first is that the subformula $\mu X.\phi$ is unsatisfiable in the sense that there is no structure $\mathcal{T}$ from the root of which Even can win at $\mu X.\phi$ in $\mathcal{T} \times \Psi(\mu X.\phi)$. Then it can be replaced with $\bot$. In the other case, $\mu X.\phi$ can be replaced with $\nu X.\phi$.

▶ **Lemma 27.** *If $\Psi(\mu X.\phi)$, a guarded formula in disjunctive form with a subformula $\mu X.\phi$, is semantically in Π_1^μ, then either there is no structure $\mathcal{T}$ such that Even wins from $(r_0, \mu X.\phi)$ in $\mathcal{T} \times \Psi(\mu X.\phi)$ where r_0 is the root of $\mathcal{T}$, or $\Psi(\mu X.\phi) = \Psi(\nu X.\phi)$.*

Proof. Assume that $\Psi(\mu X.\phi) \neq \Psi(\nu X.\phi)$ and that there is a structure $\mathcal{T}$ such that Even wins from $(r_0, \mu X.\phi)$ in $\mathcal{T} \times \Psi(\mu X.\phi)$ where r_0 is the root of $\mathcal{T}$. Since $\Psi(\mu X.\phi)$ implies $\Psi(\nu X.\phi)$ but not the other way around, then there is a structure $\mathcal{M}$ such that $\mathcal{M} \models \neg\Psi(\mu X.\phi) \wedge \Psi(\nu X.\phi)$ and $\mathcal{M}$ is well-behaved with respect to $\Psi(\nu X.\phi)$. Recall that we require for $\mathcal{M}$ to be finitely branching. We will show that for any n there is a structure $\mathcal{M}'$ which agrees with $\mathcal{M}$ up to depth n but which satisfies $\Psi(\mu X.\phi)$. Using Lemma 21, this will contradict $\Psi(\mu X.\phi) \in \Pi_1^\mu$.

For any n, we can write $\Psi(\mu X.\phi)$ as $\Psi(\overbrace{\phi...\phi}^{n}(\mu X.\phi))$, where we drop some brackets for readability, so $\phi\phi(X)$ should be understood as $\phi(\phi(X))$. Then, the structure $\mathcal{M}$ satisfies $\neg\Psi(\overbrace{\phi...\phi}^{n}(\mu X.\phi)) \wedge \Psi(\overbrace{\phi...\phi}^{n}(\top))$ since $\overbrace{\phi...\phi}^{n}(\top)$ is implied by $\nu X.\phi$. Furthermore, $\mathcal{M}$ is well-behaved for $\Psi(\overbrace{\phi...\phi}^{n}(\top))$. From Lemma 26 we know that there is a set S of states in $\mathcal{M}$ such that for each $s_i \in S$, Even loses from $(s, \neg\mu X.\phi)$ and if each $s_i \in S$ is replaced with r_0, the root of $\mathcal{T}$, to yield a new model $\mathcal{M}'$, then Ψ holds in $\mathcal{M}'$. Furthermore, since X is guarded in $\mu X.\phi$, a play can only reach $\mu X.\phi$ from $\overbrace{\phi...\phi}^{n}(\mu X.\phi)$ at depth at least n: each $s_i \in S$ is at least at depth n therefore $\mathcal{M}'$ agrees with $\mathcal{M}$ up to depth n. We have built for any n, a structure that agrees with $\mathcal{M}$, a counter-model of Ψ, up to n but satisfies Ψ. This contradicts the assumption that Ψ is in Π_1^μ, and has finite counter-proofs using Lemma 21. ◀

It now suffices to show that if a subformula $\mu X.\phi$ is unsatisfiable in the sense that there is no structure $\mathcal{T}$ from the root of which Even can win at $\mu X.\phi$ in $\mathcal{T} \times \Psi(\mu X.\phi)$, then $\mu X.\phi$ can be replaced with $\bot$. This should be intuitively justified by the idea that in no structure can Even win by playing to $\mu X.\phi$, so it is no worse for her to have $\bot$ instead.

▶ **Lemma 28.** *If there is no structure $\mathcal{T}$ rooted at t_0 such that Even wins from $(t_0, \mu X.\phi)$ in $\mathcal{T} \times \Psi(\mu X.\phi)$, then $\Psi(\mu X.\phi) = \Psi(\bot)$.*

Proof. If Even wins $\mathcal{M} \times \Psi(\mu X.\phi)$ but there is no $\mathcal{T}$ rooted at t_0 such that Even wins from $(t_0, \mu X.\phi)$, then Even's winning strategy cannot reach any position $(s, \mu X.\phi)$. Then the same strategy can be used in $\mathcal{M} \times \Psi(\bot)$ to avoid any position $(s, \bot)$. Since these two games are identical up until $\mu X.\phi$ or $\bot$ is reached, Even also wins in $\mathcal{M} \times \Psi(\bot)$. This shows $\Psi(\mu X.\phi) \implies \Psi(\bot)$. The other direction is trivial since $\bot \implies \mu X.\phi$ and L_μ is monotone. ◀

▶ **Theorem 29.** *If Ψ is a formula in guarded disjunctive form and semantically in Π_1^μ, then either $\Psi = \Psi[\bot/\mu X.\phi]$ or $\Psi[\nu X.\phi/\mu X.\phi]$ for any subformula $\mu X.\phi$ of Ψ.*

Proof. If there is no structure $\mathcal{T}$ such that Even wins from $(r_0, \mu X.\phi)$ in $\mathcal{T} \times \Psi(\mu X.\phi)$ where r_0 is the root of $\mathcal{T}$, then from the previous lemma, $\Psi = \Psi[\bot/\mu X.\phi]$. If there is such a structure, then from Lemma 27 we know that $\Psi = \Psi[\nu X.\phi/\mu X.\phi]$. ◀

▶ **Corollary 30.** *Π_1^μ and by duality Σ_1^μ are decidable.*

Proof. Any formula Ψ of L_μ can be turned into a guarded formula in disjunctive guarded form. Then, if Ψ is semantically in Π_1^μ, every occurrence of $\mu X.\phi$ can be eliminated either by replacing it with $\bot$ or $\nu X.\phi$. Hence to decide whether a formula is semantically in Π_1^μ, it is sufficient to decide whether it is equivalent to the formula where each $\mu X.\phi$ formula reachable by Even in the game for $\Psi(\mu X.\phi)$ is replaced with $\nu X.\phi$.

By duality, to decide whether a formula is semantically in Σ_1^μ it is sufficient to decide whether its negation is in Π_1^μ. If this is the case, the Π_1^μ formula can be syntactically negated to yield a formula in Σ_1^μ. ◀

5 Conclusion

We have defined syntactic transformations from L_μ into ML, Π_1^μ and Σ_1^μ which preserve meaning for formulas which are semantically, but not yet syntactically in the target class. A straight-forward corollary of this result is an alternative decision procedure for the low levels of the alternation hierarchy: to decide whether a L_μ formula is in Π_1^μ,Σ_1^μ or ML, it suffices to check whether it is equivalent to its projection into that class.

For the modal fragment of L_μ, the transformation we describe incurs a potentially exponential blow-up in the size of the formula – as such, it may be more concise to represent a formula with some fixpoints. This blow-up is however necessary since fixpoint formulas which are semantically modal can have at least subexponential modal depth.

For Π_1^μ on the other hand, assuming formulas are in guarded disjunctive form, the target formula is no larger than the original one. The transformation into guarded disjunctive form itself can incur an exponential blow-up, not least because the transformation involves distributing conjunctions over disjunctions, causing duplication.

This result is of both practical and theoretical interest since the complexity of model checking depends on the *syntactic* alternation depth of a formula, rather than the semantic one. Thus for formulas that are semantically in a low alternation class, this transformation can potentially turn an exponential model-checking procedure into a polynomial one by eliminating the exponent. By providing a concise formula in the semantic alternation class of a formula, our method provides an appealing pre-processing step for model checking.

References

1 J.C. Bradfield. The modal mu-calculus alternation hierarchy is strict. In Ugo Montanari and Vladimiro Sassone, editors, *CONCUR'96: Concurrency Theory*, volume 1119 of *Lecture Notes in Computer Science*, pages 233–246. Springer Berlin Heidelberg, 1996.

2 J.C. Bradfield and C. Stirling. Modal mu-calculi. *Handbook of modal logic*, 3:721–756, 2007.

3 E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy. In *Foundations of Computer Science, 1991. Proceedings., 32nd Annual Symposium on*, pages 368–377. IEEE, 1991.

4 D. Janin and I. Walukiewicz. Automata for the modal μ-calculus and related results. In *Proc. MFCS'95 LNCS 969*, pages 552–562, 1995.

5 D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27(3):333–354, 1983. Special Issue Ninth International Colloquium on Automata, Languages and Programming (ICALP) Aarhus, Summer 1982.

6 D. Kozen. A finite model theorem for the propositional μ-calculus. *Studia Logica*, 47(3):233–241, 1988.

7 O. Kupferman, M. Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM (JACM)*, 47(2):312–360, 2000.

8 R. Küsters and T. Wilke. Deciding the first level of the μ-calculus alternation hierarchy. In *Proc. FSTTCS 2002, LNCS 2556*, page 241–252, 2002.

9 R. Küsters and T. Wilke. Deciding the First Level of the μ-calculus Alternation Hierarchy. Technical Report 0209, Institut für Informatik und Praktische Mathematik, CAU Kiel, Germany, 2002.

10 G. Lenzi. A hierarchy theorem for the μ-calculus. In Friedhelm Meyer and Burkhard Monien, editors, *Automata, Languages and Programming*, volume 1099 of *Lecture Notes in Computer Science*, pages 87–97. Springer Berlin Heidelberg, 1996.

11 R. Mateescu. Local model-checking of modal mu-calculus on acyclic labeled transition systems. In Joost-Pieter Katoen and Perdita Stevens, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 2280 of *Lecture Notes in Computer Science*, pages 281–295. Springer Berlin Heidelberg, 2002.

12 M. Nair. A new method in elementary prime number theory. *Journal of the London Mathematical Society*, 2(3):385–391, 1982.

13 M. Otto. Eliminating recursion in the μ-calculus. In *STACS 99*, pages 531–540. Springer, 1999.

14 I. Walukiewicz. Completeness of Kozen's Axiomatisation of the Propositional μ-Calculus. *Information and Computation*, 157(1–2):142–182, 2000.

Infinite and Bi-infinite Words with Decidable Monadic Theories*

Dietrich Kuske[1], Jiamou Liu[2], and Anastasia Moskvina[2]

1 **Technische Universität Ilmenau, Germany**
2 **Auckland University of Technology, New Zealand**

Abstract

We study word structures of the form $(D, \leq, P)$ where D is either $\mathbb{N}$ or $\mathbb{Z}$, $\leq$ is a linear ordering on D and $P \subseteq D$ is a predicate on D. In particular we show:

(a) The set of recursive ω-words with decidable monadic second order theories is Σ_3-complete.
(b) We characterise those sets $P \subseteq \mathbb{Z}$ that yield bi-infinite words $(\mathbb{Z}, \leq, P)$ with decidable monadic second order theories.
(c) We show that such "tame" predicates P exist in every Turing degree.
(d) We determine, for $P \subseteq \mathbb{Z}$, the number of predicates $Q \subseteq \mathbb{Z}$ such that $(\mathbb{Z}, \leq, P)$ and $(\mathbb{Z}, \leq, Q)$ are indistinguishable.

Through these results we demonstrate similarities and differences between logical properties of infinite and bi-infinite words.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases infinite words, bi-infinite words, monadic second order logic

Digital Object Identifier 10.4230/LIPIcs.CSL.2015.472

1 Introduction

The decision problem for logical theories of linear structures and their expansions has been an important question in theoretical computer science. Büchi in [2] proved that the monadic second order theory of the linear ordering $(\mathbb{N}, \leq)$ is decidable. Expanding the structure $(\mathbb{N}, \leq)$ by unary functions or binary relations typically leads to undecidable monadic theories. Hence many works have been focusing on structures of the form $(\mathbb{N}, \leq, P)$ where P is a unary predicate. Elgot and Rabin [5] showed that for many natural unary predicates P, such as the set of factorial numbers, the set of powers of k, and the set of kth powers (for fixed k), the structure $(\mathbb{N}, \leq, P)$ has decidable monadic second order theory; on the other hand, there are structures $(\mathbb{N}, \leq, P)$ whose monadic theory is undecidable [3]. Numerous subsequent works further expanded the field [13, 4, 11, 10, 9, 8].

1. Semenov generalised periodicity to a notion of "almost periodicity". While periodicity implies that certain patterns are repeated through a fixed period, almost periodicity captures the fact that certain patterns occur before the expiration of some period. This led him to consider "recurrent structures" within an infinite word. Such a recurrent structure is captured by a certain function, which he called "indicator of recurrence". In [11], he provided a full characterisation: $(\mathbb{N}, \leq, P)$ has decidable monadic theory if and only if P is recursive and there is a recursive indicator of recurrence for P.

* This work was partially supported by the Marsden fund of New Zealand.

24th EACSL Annual Conference on Computer Science Logic (CSL 2015).
Editor: Stephan Kreutzer; pp. 472–486

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

2. Rabinovich and Thomas generalised periodicity to a notion of "uniform periodicity". Such a uniform periodicity condition is captured by a *homogeneous set* which exists by Ramsey's theorem. More precisely, a k-homogeneous set for $(\mathbb{N}, \leq, P)$ partitions the natural numbers into infinitely many finite segments that all have the same k-type. A uniformly homogeneous set specifies an ascending sequence of numbers that ultimately becomes k-homogeneous for any $k > 0$. In [9], Rabinovich and Thomas provided a full characterisation: $(\mathbb{N}, \leq, P)$ has a decidable monadic theory if and only if P is recursive and there is a recursive uniformly homogeneous set.
   Note that a recursive uniformly homogeneous set describes *how to* divide $(\mathbb{N}, \leq, P)$ such that the factors all have the same k-type. If P is recursive, this implies that the recurring k-type can be computed. A weakening of the existence of a recursive uniformly homogeneous set is therefore the requirement that one can compute a k-type such that $(\mathbb{N}, \leq, P)$ *can, in some way,* be divided. Nevertheless, Rabinovich and Thomas also showed that the monadic second order theory of $(\mathbb{N}, \leq, P)$ is decidable if and only if P is recursive and there is a "recursive type-function" (see below for precise definitions).

This paper has three general goals: The first is to compare these characterisations in some precise sense. The second is to investigate the above results in the context of *bi-infinite words*, which are structures of the form $(\mathbb{Z}, \leq, P)$. The third is to compare the logical properties of infinite words and bi-infinite words. More specifically, the paper discusses:

(a) In Section 4, we analyze the recursion-theoretical bound of the set of all computable predicates $P \subseteq \mathbb{N}$ where $(\mathbb{N}, \leq, P)$ has a decidable monadic theory. The second characterisation by Rabinovich and Thomas turns out to be a Σ_5-statement. In contrast, the characterisation by Semenov and the 1st characterisation by Rabinovich and Thomas both consist of Σ_3 statements, and hence deciding if a given $(\mathbb{N}, \leq, P)$ has decidable monadic theory is in Σ_3. We show that the problem is in fact Σ_3-complete. Hence these two characterisations are optimal in terms of their recursion-theoretical complexity.

(b) In Section 5, we then investigate which of the three characterisations can be lifted to bi-infinite words, i.e., structures of the form $(\mathbb{Z}, \leq, P)$ with $P \subseteq \mathbb{Z}$. It turns out that this is nicely possible for Semenov's characterisation and for the second characterisation by Rabinovich and Thomas, but not for their first one.

(c) If the monadic second order theory of $(\mathbb{N}, \leq, P)$ is decidable, then P is recursive. For bi-infinite words of the form $(\mathbb{Z}, \leq, P)$, this turns out not to be necessary. In Section 6, we actually show that every Turing degree contains a set $P \subseteq \mathbb{Z}$ such that the monadic second order theory of $(\mathbb{Z}, \leq, P)$ is decidable.

(d) The final Section 7 investigates how many bi-infinite words are indistinguishable from $(\mathbb{Z}, \leq, P)$. It turns out that this depends on the periodicity properties of P: if P is periodic, there are only finitely many equivalent bi-infinite words, if P is recurrent and non-periodic, there are $2^{\aleph_0}$ many, and if P is not recurrent, then there are $\aleph_0$ many.

2 Preliminaries

2.1 Words

We use $\mathbb{N}$, $\widetilde{\mathbb{N}}$ and $\mathbb{Z}$ to denote the set of natural numbers (including 0), negative integers (not containing 0), and integers, respectively. A *finite word* is a mapping $u\colon \{0, 1, \ldots, n-1\} \to \{0, 1\}$ with $n \in \mathbb{N}$, it is usually written $u(0)u(1)u(2)\cdots u(n-1)$. The set of positions of u is $\{0, 1, \ldots, n-1\}$, its length $|u|$ is n. The unique finite word of length 0 is denoted ε. The set of all (resp. non-empty) finite words is $\{0, 1\}^*$ (resp. $\{0, 1\}^+$). An *ω-word* is a mapping $\alpha\colon \mathbb{N} \to \{0, 1\}$; it is usually written as the sequence $\alpha(0)\alpha(1)\alpha(2)\cdots$. Its set of positions

is $\mathbb{N}$; $\{0,1\}^\omega$ is the set of ω-words. An *ω^*-word* is a mapping $\alpha\colon \widetilde{\mathbb{N}} \to \{0,1\}$; it is usually written as the sequence $\cdots\alpha(-3)\alpha(-2)\alpha(-1)$. Its set of positions is $\widetilde{\mathbb{N}}$ and $\{0,1\}^{\omega^*}$ is the set of ω^*-words. Finally, a *bi-infinite word* ξ is a mapping from $\mathbb{Z}$ into $\{0,1\}$, written as the sequence $\cdots\xi(-2)\xi(-1)\xi(0)\xi(1)\xi(2)\cdots$ (this notation has to be taken with care since, e.g., the bi-infinite words $\xi_i\colon \mathbb{Z} \to \{0,1\}\colon n \mapsto (|n|+i) \bmod 2$ with $i \in \{0,1\}$ are both described as $\cdots 0101010\cdots$, but they are different). The set of positions of a bi-infinite word is $\mathbb{Z}$. When saying "word", we mean "a finite, an ω-, an ω^*- or a bi-infinite word", "infinite word" means "ω- or ω^*-word".

The concatenation uv of two finite words u, v has its usual meaning. More generally, and in a similar way, we can also concatenate a finite or ω^*-word u and a finite or ω-word v giving rise to some word uv. Similarly, we can concatenate infinitely many finite words u_i giving an ω-word $u_0u_1u_2\cdots$, an ω^*-word $\cdots u_{-2}u_{-1}u_0$, and a bi-infinite word $\cdots u_{-2}u_{-1}u_0u_1u_2\cdots$ (where the position 0 is the first position of u_0). As usual, u^ω denotes the ω-word $uuuu\cdots$ for $u \in \{0,1\}^+$, analogously, $u^{\omega^*} = \cdots uuu$.

Let w be some word and i, j be two positions with $i \leq j$. Then we write $w[i,j]$ for the finite word $w(i)w(i+1)\cdots w(j) \in \{0,1\}^+$. A finite word u is a *factor* of w if $u = w[i,j]$ for some i, j or if u is the empty word ε. The set of factors of w is $F(w)$. If w is an ω- or a bi-infinite word, then $w[i,\infty)$ is the ω-word $w(i)w(i+1)w(i+2)\cdots$. If w is an ω^*- or a bi-infinite word, then $w(-\infty,i]$ is the ω^*-word $\cdots w(i-2)w(i-1)w(i)$. A bi-infinite word β is *recurrent* if for all $u \in F(\beta)$ and all $i \in \mathbb{Z}$, $u \in F(\beta[i,\infty)) \cap F(\beta(-\infty,i])$.

Let u be some finite word. Then u^R is the *reversal* of u, i.e., the finite word of length $|u|$ with $u^R(i) = u(|u|-i-1)$ for all $0 \leq i < |u|$. The reversal of an ω-word (resp. ω^*-word) α is the ω^*-word (resp. ω-word) α^R with $\alpha^R(i) = \alpha(-i-1)$ for all positions i. Finally, the reversal of a bi-infinite word ξ is the bi-infinite word ξ^R with $\xi^R(i) = \xi(-i)$ for all $i \in \mathbb{Z}$.

2.2 Logic

With any word w, we associate a relational structure $M_w = (D, \leq, P)$ where $D \subseteq \mathbb{Z}$ is the set of positions of w, $\leq$ is the restriction of the natural linear order on $\mathbb{Z}$ to D, and $P = \{n \in D \mid w(n) = 1\} = w^{-1}(1)$. Structures of this form are called *labeled linear orders*. The word w is *recursive* (resp. *recursively enumerable*) if so is the set P.

We use the standard logical system over the signature of labeled linear orders. Hence first order logic FO has relational symbols $\leq$ and P. The monadic second order logic MSO extends FO by allowing unary second order variables $X, Y, \ldots$, their corresponding atomic predicates (e.g. $X(y)$), and quantification over set variables. By Sent, we denote the set of sentences of the logic MSO. For a word w and an MSO-sentence φ, we write $w \models \varphi$ for "the sentence φ holds in the relational structure M_w". The MSO-*theory* of the word w is the set $\mathsf{MTh}(M)$ of all MSO-sentences φ that are true in w.

▶ **Example 1.** Let $n \in \mathbb{N}$ and consider the following formula:

$$\varphi(x,y) = \exists X\colon \forall z\colon (X(z) \Leftrightarrow z = x \vee (x < z \wedge X(z-n))) \wedge X(y)$$

If w is a word with positions i, j, then $w \models \varphi(i,j)$ if and only if $i \leq j$ and $n \mid j-i$.

With any MSO-formula φ, we associate its *quantifier rank* $\mathsf{qr}(\varphi) \in \mathbb{N}$: the atomic formulas have quantifier rank 0; $\mathsf{qr}(\varphi_1 \wedge \varphi_2) = \mathsf{qr}(\varphi_1 \vee \varphi_2) = \max\{\mathsf{qr}(\varphi_1), \mathsf{qr}(\varphi_2)\}$; $\mathsf{qr}(\neg\varphi) = \mathsf{qr}(\varphi)$; and $\mathsf{qr}(\exists X\colon \varphi) = \mathsf{qr}(\forall X\colon \varphi) = \mathsf{qr}(\varphi_1) + 1$ where X is a first- or second-order variable.

▶ **Definition 2.** Let $k \in \mathbb{N}$. Two words w_1 and w_2 are *k-equivalent* (denoted $w_1 \equiv_k w_2$) if $w_1 \models \varphi$ iff $w_2 \models \varphi$ for all MSO-sentences φ with $\mathsf{qr}(\varphi) \leq k$. Equivalence classes of this equivalence relation are called *k-types*. The words w_1 and w_2 are MSO-*equivalent* (denoted $w_1 \equiv w_2$) if $w_1 \equiv_k w_2$ for all $k \in \mathbb{N}$. Equivalence classes of $\equiv$ are called *types*.

Let $k \geq 2$ and u, v be two words with $u \equiv_k v$. If u is finite, then it satisfies the sentence $(\exists x \forall y\colon x \leq y) \wedge (\exists x \forall y\colon x \geq y)$. Consequently, also v is finite. Analogously, u is an ω-word iff v is an ω-word etc. We will therefore speak of a "k-type of finite words" when we mean a k-type that contains some finite word (and analogously for ω-, ω^*-, bi-infinite words etc).

Often, we will use the following known results without mentioning them again. They follow from the well-understood relation between MSO and automata (cf. [15, 6]).

▶ **Theorem 3.**

1. *Let $k \geq 2$.*
   - *For any ω-word (ω^*-word) α, there exist finite words x and y with $xy \equiv_k x$ ($yx \equiv_k x$), $yy \equiv_k y$ and $\alpha \equiv_k xy^\omega$ ($\alpha \equiv_k y^{\omega^*}x$). Any such pair (x, y) is* a representative of the k-type of α.
   - *For any bi-infinite word ξ, there exist finite words x, y and z with $xy \equiv_k yz \equiv_k y$, $xx \equiv_k x$, $zz \equiv_k z$, and $\xi \equiv_k x^{\omega^*}yz^\omega$. Any such triple (x, y, z) is* a representative of the k-type of ξ.
2. *The following sets are decidable:*
   - $\{\varphi \in \mathsf{Sent} \mid \forall u \in \{0,1\}^*\colon u \models \varphi\}$ *and* $\{(u, \varphi) \mid u \in \{0,1\}^*, \varphi \in \mathsf{Sent}, u \models \varphi\}$
   - $\{(u, v, \varphi) \mid u, v \in \{0,1\}^*, v \neq \varepsilon, \varphi \in \mathsf{Sent}, uv^\omega \models \varphi\}$
   - $\{(u, v, w, \varphi) \mid u, v, w \in \{0,1\}^*, u, w \neq \varepsilon, \varphi \in \mathsf{Sent}, u^{\omega^*}vw^\omega \models \varphi\}$
   - $\{(u, v, k) \mid u, v \in \{0,1\}^*, k \in \mathbb{N}, u \equiv_k v\}$. *This means in particular that it is decidable whether u and v represent the same k-type of finite words.*
   - *Similarly, it is decidable whether two pairs of finite words represent the same k-type of ω-words (of ω^*-words, resp). It is also decidable whether two triples of finite words represent the same k-type of bi-infinite words.*
3. *If $u, v \in \{0,1\}^* \cup \{0,1\}^{\omega^*}$ and $u', v' \in \{0,1\}^* \cup \{0,1\}^\omega$ with $u \equiv_k v$ and $u' \equiv_k v'$, then $uu' \equiv_k vv'$. From representatives of the k-types of u and v, one can compute a representative of the k-type of uv.*
4. *If $u_i, v_i \in \{0,1\}^+$ with $u_i \equiv_k v_i$ for all $i \in \mathbb{Z}$, then we have*

$$u_0u_1\cdots \equiv_k v_0v_1\cdots, \text{ and } \cdots u_{-1}u_0 \equiv_k \cdots v_{-1}v_0, \text{ and } \cdots u_{-1}u_0u_1\cdots \equiv_k \cdots v_{-1}v_0v_1\cdots$$

5. *If u is a finite or ω^*-word and v is a finite or ω-word such that $\mathsf{MTh}(u)$ and $\mathsf{MTh}(v)$ are both decidable, then $\mathsf{MTh}(uv)$ is decidable [12].*

2.3 Recursion theoretic notions

This paper makes use of standard notions in recursion theory; the reader is referred to [14] for a thorough introduction. We assume a canonical effective enumeration $\Phi_0, \Phi_1, \Phi_2, \ldots$ of all partial recursive functions on the natural numbers. The set W_e is the domain $\mathsf{dom}(\Phi_e)$ and is the *eth recursively enumerable set*. Let TOT be the set $\{e \in \mathbb{N} \mid \Phi_e \text{ is total}\}$ and REC be the set $\{e \in \mathbb{N} \mid W_e \text{ is decidable}\}$.

A set $A \subseteq \mathbb{N}$ belongs to the level Π_2 of the arithmetical hierarchy if there exists a decidable set $P \subseteq \mathbb{N}^{m+n+1}$ such that A is the set of natural numbers a satisfying $\forall x_1, \ldots, x_m \exists y_1, \ldots y_n\colon P(a, \bar{x}, \bar{y})$. A set $B \subseteq \mathbb{N}$ is Π_2-*hard* if, for every $A \in \Pi_2$, there exists a m-reduction from A to B; the set B is Π_2-*complete* if, in addition, $B \in \Pi_2$. Similarly, $A \subseteq \mathbb{N}$ belongs to Σ_3 if there exists a decidable set $P \subseteq \mathbb{N}^{\ell+m+n+1}$ such that A is the set of natural numbers a satisfying $\exists x_1, \ldots, x_\ell \forall y_1, \ldots, y_m \exists z_1, \ldots z_n\colon P(a, \bar{x}, \bar{y}, \bar{z})$. The notions Σ_3-hard and Σ_3-complete are defined similarly. For our purposes, it is important that the set TOT is Π_2-complete and the set REC is Σ_3-complete [14].

3 When is the MSO-theory of an ω-word decidable?

In this section, we recall the answers by Semenov [11] and by Rabinovich and Thomas [9]. Semenov defined a form of *"periodic words"* in which words from certain regular sets recur.

▶ **Definition 4.** Let α be some ω-word. An *indicator of recurrence* for α is a function $\mathrm{rec}\colon \mathsf{Sent} \to \mathbb{N} \cup \{\top\}$ such that, for every MSO-sentence φ, the following hold:

- if $\mathrm{rec}(\varphi) = \top$, then $\forall k\, \exists j \geq i \geq k\colon \alpha[i,j] \models \varphi$
- if $\mathrm{rec}(\varphi) \neq \top$, then $\forall j \geq i \geq \mathrm{rec}(\varphi)\colon \alpha[i,j] \models \neg\varphi$

▶ **Theorem 5** (Semenov's Characterisation [11]). *Let α be an ω-word. Then $\mathsf{MTh}(\alpha)$ is decidable if and only if the ω-word α is recursive and there exists a recursive indicator of recurrence for α.*

Note that an ω-word can have many recursive indicators of recurrence: if rec is such an indicator, then so is $\varphi \mapsto 2 \cdot \mathrm{rec}(\varphi)$.

Two other characterisations are given by Rabinovich and Thomas in [9]. The idea is to decompose an infinite word into infinitely many finite sections all of which (except possibly the first one) have the same k-type.

▶ **Definition 6.** Let $\alpha \in \{0,1\}^\omega$, $u, v \in \{0,1\}^+$, $k \in \mathbb{N}$, and $H \subseteq \mathbb{N}$ be infinite.

- The set H is a *k-homogeneous factorisation of α into (u,v)* if $\alpha[0, i-1] \equiv_k u$ and $\alpha[i, j-1] \equiv_k v$ for all $i, j \in H$ with $i < j$. The set H is *k-homogeneous for α* if it is a k-homogeneous factorisation of α into some finite words (u,v).
- Let $H = \{h_i \mid i \in \mathbb{N}\}$ with $h_0 < h_1 < \dots$. The set H is *uniformly homogeneous for α* if, for all $k \in \mathbb{N}$, the set $\{h_i \mid i \geq k\}$ is k-homogeneous for α.

As with indicators of recurrence, any ω-word has many uniformly homogeneous sets: the existence of at least one follows by a repeated and standard application of Ramsey's theorem, and there are infinitely many since any infinite subset of a uniformly homogeneous set is again uniformly homogeneous.

▶ **Theorem 7** (1st Rabinovich-Thomas' Characterisation [9]). *Let α be an ω-word. Then $\mathsf{MTh}(\alpha)$ is decidable if and only if the ω-word α is recursive and there exists a recursive uniformly homogeneous set for α.*

Suppose $h_0 < h_1 < h_2 < \dots$ is an enumeration of some uniformly homogeneous set for α. This sequence determines finite words u_k and v_k such that $w \equiv_k u_k(v_k)^\omega$, $u_k v_k \equiv_k u_k$, and $v_k v_k \equiv_k v_k$: simply set $u_k = \alpha[0, h_k - 1]$ and $v_k = \alpha[h_k, h_{k+1} - 1]$. If the ω-word α is recursive, we can therefore, from $k \in \mathbb{N}$, compute a representative of the k-type of α.

▶ **Definition 8.** Let α be some ω-word and $\mathrm{tp}\colon \mathbb{N} \to \{0,1\}^+ \times \{0,1\}^+$. The function tp is a *type-function* if, for all $k \in \mathbb{N}$, α has a k-homogeneous factorisation into $\mathrm{tp}(k) = (u,v)$.

Let tp be a type-function for the ω-word α and let $k \in \mathbb{N}$. Then there exists a k-homogeneous factorisation H of α into $\mathrm{tp}(k) = (u,v)$. Let $H = \{h_0 < h_1 < h_2 < \dots\}$. Then we have $\alpha = \alpha[0, h_0 - 1]\, \alpha[h_0, h_1 - 1]\alpha[h_1, h_2 - 1] \cdots \equiv_k uv^\omega$. Furthermore, $v \equiv_k \alpha[h_0, h_2 - 1] = \alpha[h_0, h_1 - 1]\alpha[h_1, h_2 - 1] \equiv_k vv$. Consequently, $\mathrm{tp}(k)$ is a representative of the k-type of α.

▶ **Theorem 9** (2nd Rabinovich-Thomas' Characterisation [9]). *Let α be an ω-word. Then $\mathsf{MTh}(\alpha)$ is decidable if and only if α has a recursive type-function.*

Note that, differently from Thm. 7 this theorem does not mention that α is recursive. But this recursiveness is implicit: Let tp be a recursive type-function and $k \in \mathbb{N}$. Then one can write a FO sentence of quantifier-depth $k+2$ expressing that $\alpha(k) = 1$. Let $\mathrm{tp}(k+2) = (u,v)$. Then $\alpha \equiv_{k+2} uv^\omega$ implies $\alpha(k) = uv^k(k)$, hence $\alpha(k)$ is computable from k.

4 How hard is it to tell if the MSO-theory of an ω-word is decidable?

In this section, we determine the recursion-theoretical complexity of the question whether the MSO-theory of a recursive ω-word is decidable. Technically, we will consider the following two sets:

$$\mathsf{DecTh}_{\mathbb{N}}^{\mathsf{MSO}} = \{e \in \mathsf{REC} \mid \mathsf{MTh}(\mathbb{N}, \leq, W_e) \text{ is decidable}\} \qquad \mathsf{UndecTh}_{\mathbb{N}}^{\mathsf{MSO}} = \mathsf{REC} \setminus \mathsf{DecTh}_{\mathbb{N}}^{\mathsf{MSO}}$$

Recall that $W_e \subseteq \mathbb{N}$ denotes the e^{th} recursively enumerable set.

But first note the following: Let α be some recursive word. Then, by Büchi's and McNaughton's theorems, $\mathsf{MTh}(\alpha)$ is decidable iff the set of deterministic parity automata accepting α is decidable. Recall that "the deterministic parity automaton no. n accepts α" (where we assume any computable enumeration of all deterministic parity automata) is a Boolean combination of Σ_2-statements, cf. [15, Prop. 5.3]. It follows that $e \in \mathsf{DecTh}_{\mathbb{N}}^{\mathsf{MSO}}$ if and only if the following holds:

$$\exists f \in \mathsf{TOT}\, \forall n\colon \Phi_f(n) = 1 \Leftrightarrow \text{the deterministic parity automaton no. } n \text{ accepts } (\mathbb{N}, \leq, W_e)$$

Hence $\mathsf{DecTh}_{\mathbb{N}}^{\mathsf{MSO}}$ belongs to Σ_4. The following lemma improves this by one level in the arithmetical hierarchy:

▶ **Lemma 10.** *The set* $\mathsf{DecTh}_{\mathbb{N}}^{\mathsf{MSO}}$ *belongs to* Σ_3.

We present two proofs of this lemma, one based on the first Rabinovich-Thomas characterisation, the second one based on the Semenov characterisation.

Proof. (based on Thm. 7) Let α be some recursive ω-word. Recall that a set $H \subseteq \mathbb{N}$ is infinite and recursive if there exists a total computable and strictly monotone function f such that $H = \{f(n) \mid n \in \mathbb{N}\}$. Now consider the following:

$$\begin{aligned}\exists e\, \forall k, i, j, i', j'\colon\ & e \in \mathsf{TOT} \wedge\ (i < j \Rightarrow \Phi_e(i) < \Phi_e(j)) \wedge \\ & (k \leq i < j \wedge k \leq i' < j' \Rightarrow \alpha[\Phi_e(i), \Phi_e(j) - 1] \equiv_k \alpha[\Phi_e(i'), \Phi_e(j') - 1])\end{aligned}$$

It expresses that there exists a total recursive function (namely Φ_e) that is strictly monotone. Its image then consists of the numbers $\Phi_e(0) < \Phi_e(1) < \Phi_e(2) < \dots$ The last line expresses that this image is uniformly homogeneous for α. Hence this statement says that there exists a recursive uniformly homogeneous set for α, i.e., that $\mathsf{MTh}(\alpha)$ is decidable by Thm. 7.

From $k, i, i', j, j' \in \mathbb{N}$ with $k \leq i < j$, and $k \leq i' < j'$ we can compute the finite words $\alpha[\Phi_e(i), \Phi_e(j) - 1]$ and $\alpha[\Phi_e(i'), \Phi_e(j') - 1]$ since α is recursive. Hence it is decidable whether $\alpha[\Phi_e(i), \Phi_e(j) - 1] \equiv_k \alpha[\Phi_e(i'), \Phi_e(j') - 1]$. The whole statement is in Σ_3 as $\mathsf{TOT} \in \Pi_2$. ◀

Proof. (based on Thm. 5) We enumerate the set Sent of MSO-sentences in any effective way as $\varphi_0, \varphi_1, \dots$. Let $e \in \mathsf{TOT}$. Then the function rec: $\mathsf{Sent} \to \mathbb{N}$: $\varphi_i \mapsto \Phi_e(i)$ is an indicator of recurrence for the ω-word α if and only if the following holds for all $\varphi \in \mathsf{Sent}$

$$(\mathrm{rec}(\varphi) \neq \top \Rightarrow \forall k \geq j \geq \mathrm{rec}(\varphi)\colon \alpha[j, k] \models \neg\varphi)\ \wedge\ (\mathrm{rec}(\varphi) = \top \Rightarrow \forall j \exists \ell \geq k \geq j\colon \alpha[k, \ell] \models \varphi)$$

Given the definition of rec, this is equivalent to requiring (for all $i \in \mathbb{N}$)

$$(\Phi_e(i) \neq \top \Rightarrow \forall k \geq j \geq \Phi_e(i)\colon \alpha[j, k] \models \neg\varphi_i)\ \wedge\ (\Phi_e(i) = \top \Rightarrow \forall j \exists \ell \geq k \geq j\colon \alpha[k, \ell] \models \varphi_i)$$

If α is recursive, this is a Π_2-statement. Prefixing it with $\exists e \in \mathsf{TOT}\, \forall i$ yields a Σ_3-statement that expresses the existence of a recursive indicator of recurrence. ◀

▶ Remark. From the 2nd characterisation by Rabinovich and Thomas (Thm. 9), we can only infer that $\mathsf{DecTh}_{\mathbb{N}}^{\mathsf{MSO}}$ is in Σ_5: Let α be some recursive ω-word and $u, v \in \{0,1\}^+$. Then, by the proof of [9, Prop. 7], there exists a k-homogeneous factorisation of α into (u, v), if the following Σ_3-statement $\varphi(u, v)$ holds: $\exists x \forall y \exists z, z' \colon (\alpha[0, x-1] \equiv_k u \wedge y < z < z' \wedge \alpha[x, z-1] \equiv_k \alpha[z, z'-1] \equiv_k v)$. Hence the function $\mathrm{tp} \colon \mathbb{N} \to \{0,1\}^+ \times \{0,1\}^+$ is a type-function if the Π_4-statment $\forall k \in \mathbb{N} \colon \varphi(\mathrm{tp}(k))$ holds. Consequently, there is a recursive type-function if we have $\exists e \colon e \in \mathsf{TOT} \wedge \forall k \colon \varphi(\Phi_e(k))$ which is a Σ_5-statement.

The above raises the natural question whether these characterisations are "optimal". Namely, if one can separate $\mathsf{DecTh}_{\mathbb{N}}^{\mathsf{MSO}}$ from $\mathsf{UndecTh}_{\mathbb{N}}^{\mathsf{MSO}}$ using a simpler statement. We now prepare a negative answer to this last question (which is an affirmative answer to the optimality question posed first).

We now construct an m-reduction from the set REC to any separator of $\mathsf{DecTh}_{\mathbb{N}}^{\mathsf{MSO}}$ and $\mathsf{UndecTh}_{\mathbb{N}}^{\mathsf{MSO}}$: Let $e \in \mathbb{N}$. One can compute $f \in \mathbb{N}$ such that Φ_f is total and injective and $\{\Phi_f(i) \mid i \in \mathbb{N}\} = \{2a \mid a \in W_e\} \cup (2\mathbb{N}+1)$. For $i \in \mathbb{N}$, set $x_i = 2^{\Phi_f(i)} \times \prod_{0 \le j \le i}(2j+1)$ and consider the ω-word $\alpha_e = 10^{x_0}10^{x_1}10^{x_2}\cdots$. Since Φ_f is total, this ω-word is recursive.

▶ **Lemma 11.** *Let $e \in \mathbb{N}$. The MSO-theory of the ω-word α_e is decidable if and only if the e^{th} recursively enumerable set W_e is recursive, i.e., $e \in \mathsf{REC}$.*

Proof. First suppose that the MSO-theory of α_e is decidable. For $a \in \mathbb{N}$, we have $a \in W_e$ iff there exists $i \ge 0$ with $2a = \Phi_f(i)$ iff there exists $i \ge 0$ such that 2^{2a} is the greatest power of 2 that divides x_i. Consequently, $a \in W_e$ if the ω-word α_e satisfies

$$\exists x, y \in P \colon (x < y \wedge \forall z \colon (x < z < y \Rightarrow z \notin P)) \ \wedge \ \left(2^{2a} \mid y - x - 1 \wedge 2^{2a+1} \not| y - x - 1\right) \quad (1)$$

Recall that $n \mid y - x - 1$ is expressible by an MSO-formula. Since validity in α_e of the resulting MSO-sentence is decidable, the set W_e is recursive.

Conversely, let W_e be recursive. To show that the MSO-theory of α_e is decidable, let φ be some MSO-sentence. Let $k = \mathsf{qr}(\varphi)$ be the quantifier-rank of φ. To decide whether $\alpha_e \models \varphi$, we proceed as follows:

- Using standard semigroup arguments, compute $\ell > 0$ such that $0^\ell \equiv_k 0^{2\ell}$ and determine $a, b \in \mathbb{N}$ with $\ell = 2^a(2b+1)$.
- Compute $i \ge b$ such that $\Phi_f(j) > a$ for all $j > i$: to this aim, first determine $A = \{n \le a \mid n \in W_e \text{ or } a \text{ odd}\}$ which is possible since W_e is decidable. Then compute the least $i \ge b$ such that $A \subseteq \{\Phi_f(j) \mid j \le i\}$. Since Φ_f is injective, $\Phi_f(j) > a$ for all $j > i$.
- Decide whether $10^{x_0}10^{x_1}\ldots10^{x_i}(10^\ell)^\omega$ satisfies φ which is possible since this ω-word is ultimately periodic.

Let $j > i$. Then $\Phi_f(j) > a$ and $j > i \ge a$ imply that x_j is a multiple of ℓ. Thus $0^{x_j} \equiv_k 0^\ell$. We therefore obtain $\alpha_e \equiv_k 10^{x_1}10^{x_2}\cdots10^{x_i}(10^\ell)^\omega$. Hence the above algorithm is correct. ◀

Lemmas 11 and 10 imply that the problem of deciding whether a recursive ω-word has a decidable MSO-theory is Σ_3-complete:

▶ **Theorem 12.** - $\mathsf{DecTh}_{\mathbb{N}}^{\mathsf{MSO}}$ *is in Σ_3.*
- *Any set containing $\mathsf{DecTh}_{\mathbb{N}}^{\mathsf{MSO}}$ and disjoint from $\mathsf{UndecTh}_{\mathbb{N}}^{\mathsf{MSO}}$ is Σ_3-hard.*

▶ Remark. Thm. 7 also holds for the weaker logics FO and $\mathsf{FO}{+}\mathsf{MOD}$ that extends FO by modulo-counting quantifiers [9]. Consequently, Lemma 10 also holds, *mutatis mutantis*, for these logics.

Conversely, Lemma 11 also holds for $\mathsf{FO}{+}\mathsf{MOD}$ since (1) is easily expressible in this logic. To also handle FO, replace the definition of x_i by $x_i = \Phi_f(j)$. A similar argument as in Lemma 11 proves that W_e is recursive iff the ω-word α_e obtained this way has a decidable FO-theory. Thus, Thm. 12 also holds for the logics FO and $\mathsf{FO}{+}\mathsf{MOD}$.

5 When is the MSO-theory of a bi-infinite word decidable?

In this section, we investigate whether the characterisations from Theorems 5, 7, and 9 can be lifted from ω- to bi-infinite words.

5.1 A characterisation à la Semenov

▶ **Definition 13.** Let ξ be a bi-infinite word. A pair of functions $(\mathrm{rec}_\leftarrow, \mathrm{rec}_\rightarrow)$ with $\mathrm{rec}_\leftarrow, \mathrm{rec}_\rightarrow \colon \mathsf{Sent} \to \mathbb{Z} \cup \{\top\}$ is an *indicator of recurrence for* ξ if for any $\varphi \in \mathsf{Sent}$:

- if $\mathrm{rec}_\leftarrow(\varphi) = \top$, $\forall k \in \mathbb{Z}\, \exists i \le j \le k \colon \xi[i,j] \models \varphi$; otherwise, $\forall i \le j \le \mathrm{rec}_\leftarrow(\varphi) \colon \xi[i,j] \models \neg\varphi$
- if $\mathrm{rec}_\rightarrow(\varphi) = \top$, $\forall k \in \mathbb{Z}\, \exists j \ge i \ge k \colon \xi[i,j] \models \varphi$; otherwise, $\forall j \ge i \ge \mathrm{rec}_\rightarrow(\varphi) \colon \xi[i,j] \models \neg\varphi$

A bi-infinite word ξ "consists" of an ω^*-word $\xi_\leftarrow$ and an ω-word $\xi_\rightarrow$. Then, roughly speaking, an indicator of recurrence for the *bi-infinite* word ξ consists of a pair of indicators of recurrence, one for $\xi_\leftarrow$ and one for $\xi_\rightarrow$. Therefore, the following is similar to Thm. 5.

▶ **Theorem 14.** *Let ξ be a bi-infinite word. Then $\mathsf{MTh}(\xi)$ is decidable if and only if ξ has a recursive indicator of recurrence and the bi-infinite word ξ is recursive or recurrent.*

This theorem is an immediate consequence of Propositions 15 and 16 below. If ξ is non-recurrent, there is a finite word u that has a leftmost or a rightmost occurrence in ξ, say at a position $x \in \mathbb{Z}$. Then x is definable in MSO. Consequently, also the position 0 is definable. This allows one to reduce the decidability of $\mathsf{MTh}(\xi)$ to the decidability of both $\mathsf{MTh}(\xi(-\infty,-1])$ and $\mathsf{MTh}(\xi[0,\infty))$. Hence Prop. 15 is a consequence of Thm. 5.

▶ **Proposition 15.** *Let ξ be a non-recurrent bi-infinite word. Then $\mathsf{MTh}(\xi)$ is decidable if and only if ξ has a recursive indicator of recurrence and the bi-infinite word ξ is recursive.*

▶ **Proposition 16.** *Let ξ be a recurrent bi-infinite word. Then $\mathsf{MTh}(\xi)$ is decidable if and only if ξ has a recursive indicator of recurrence.*

Proof. First suppose $\mathsf{MTh}(\xi)$ is decidable. We have to construct a recursive indicator of recurrence $(\mathrm{rec}_\leftarrow, \mathrm{rec}_\rightarrow)$ for ξ. Let $\varphi \in \mathsf{Sent}$. Set $\mathrm{rec}_\leftarrow(\varphi) = \mathrm{rec}_\rightarrow(\varphi) = \top$ if there exist integers $i \le j$ with $\xi[i,j] \models \varphi$, otherwise set $\mathrm{rec}_\leftarrow(\varphi) = \mathrm{rec}_\rightarrow(\varphi) = 0$.

It remains to be shown that these functions are recursive and that they form an indicator of recurrence. Regarding the recursiveness, note that there are $i \le j$ with $\xi[i,j] \models \varphi$ iff $\xi \models \exists x, y \colon x \le y \wedge \varphi_{x,y}$ where $\varphi_{x,y}$ is obtained form φ by restricting all quantifiers to the interval $[x,y]$. Since $\mathsf{MTh}(\xi)$ is decidable, the functions $\mathrm{rec}_\leftarrow$ and $\mathrm{rec}_\rightarrow$ are recursive.

Next we show that $(\mathrm{rec}_\leftarrow, \mathrm{rec}_\rightarrow)$ is an indicator of recurrence for ξ: If $\mathrm{rec}_\leftarrow(\varphi) = \top$, then (by the definition of $\mathrm{rec}_\leftarrow$) there are $i \le j$ with $\xi[i,j] \models \varphi$. Since ξ is recurrent, it follows that there are arbitrary small and large integers $a \le b$ with $\xi[a,b] = \xi[i,j] \models \varphi$. If, in the other case, $\mathrm{rec}_\leftarrow(\varphi) = 0$, then there are no integers $i \le j$ with $\xi[i,j] \models \varphi$, in particular, there are no integers $i \le j \le \mathrm{rec}_\leftarrow(\varphi)$ with $\xi[i,j] \models \varphi$.

Conversely, suppose $(\mathrm{rec}_\leftarrow, \mathrm{rec}_\rightarrow)$ is a recursive indicator of recurrence for ξ. Then, for $\varphi \in \mathsf{Sent}$, we can decide whether there are integers $i \le j$ with $\xi[i,j] \models \varphi$ (since ξ is recurrent, this is the case if and only if $\mathrm{rec}_\leftarrow(\varphi) = \top$). In [1, Thm. 3.1(2)] and in [11, 7], it is stated that then $\mathsf{MTh}(\xi)$ is decidable (a proof can be extracted from [6, Section IX.6]). ◀

Thm. 14 connects the decidability of the MSO theory of a recurrent bi-infinite word ξ with a decidability question on its set of factors $F(\xi)$. It follows that, if $\mathsf{MTh}(\xi)$ is decidable, then $F(\xi)$ is decidable. We now show that the converse implication does not hold.

▶ **Lemma 17.** *A set of finite words F containing at least one non-empty word is the factor set of a recurrent bi-infinite word if and only if it satisfies the following conditions:*

(a) *If $uvw \in F$, then $v \in F$.*
(b) *For any $u, w \in F$, there is a word $v \in F$ such that $uvw \in F$*

Proof. Necessity of (a) and (b) is obvious. So suppose $F \subseteq \{0,1\}^*$ contains at least one non-empty word u and satisfies (a) and (b). We construct a bi-infinite recurrent word ξ such that $F(\xi) = F$. Since F is non-empty, (b) implies that F is infinite. Let $F = \{u_i \mid i \in \mathbb{N}\}$. Inductively, we define two sequences $(x_i)_{i>0}$ and $(y_i)_{i>0}$ of words from F such that, for all $i \in \mathbb{N}$, the finite word $w_i = u_i x_i u_{i-1} x_{i-1} \dots u_1 x_1 u_0 y_1 u_1 y_2 u_2 \dots y_i u_i$ belongs to F.

Let $i > 0$ and suppose we already defined the words x_j and y_j for $j < i$ such that $w_{i-1} \in F$. Then, by (b), there exists $x_i \in F$ such that $u_i x_i w_{i-1} \in F$. Again by (b), there exists $y_i \in F$ such that $u_i x_i w_{i-1} y_i u_i \in F$. Now set $\xi = \cdots u_3 x_3\, u_2 x_2\, u_1 x_1\, u_0\, y_1 u_1\, y_2 u_2\, y_3 u_3 \cdots$. Let $v \in \{0,1\}^*$ be some factor of ξ. Then there is $i \in \mathbb{N}$ such that v is a factor of w_i. Since $w_i \in F$, condition (a) implies $v \in F$. Hence $F(\xi) = F$.

Now let $v \in F(\xi) = F$. By (b), there are infinitely many $i \in \mathbb{N}$ such that v is a factor of u_i. Hence ξ is recurrent. ◀

▶ **Theorem 18.** *There exists a recurrent bi-infinite word ξ whose set of factors is decidable, but $\mathsf{MTh}(\xi)$ is undecidable.*

Proof. Let $f\colon \mathbb{N} \to \mathbb{N}$ be some recursive and total function such that $\{f(i) \mid i \in \mathbb{N}\}$ is not recursive. Let $F \subseteq \{0,1\}^*$ be the set of all finite words u with the following property: If $10^{2i+1}10^{2j}1$ is a factor of u, then $j = f(i)$. This set is clearly recursive, contains a non-empty word, and satisfies conditions (a) and (b) from Lemma 17. Hence there exists a bi-infinite word ξ with $F(\xi) = F$. For $j \in \mathbb{N}$, consider the following sentence:

$$\exists x < y\colon P(x) \wedge P(y + 2j) \wedge \neg 2 \mid y - x - 1 \wedge \forall z\colon (x < z < y + 2j \wedge P(z) \to z = y)$$

It expresses that the language $1(00)^*010^{2j}1$ contains a factor of ξ. But this is the case iff it contains a factor of some word from F iff there exists $i \in \mathbb{N}$ with $j = f(i)$. Since this is undecidable, the MSO-theory of ξ is undecidable by Thm. 14. ◀

5.2 A characterisation à la Rabinovich-Thomas I

We return to the question when the MSO-theory of a recurrent bi-infinite word is decidable. We will see that Thm. 7 naturally extends to *recursive* bi-infinite words. We will then demonstrate that it does not extend to non-recursive bi-infinite words.

▶ **Definition 19.** Let $\xi \in \{0,1\}^{\mathbb{Z}}$, $u, v, w \in \{0,1\}^+$, $k \in \mathbb{N}$, and let $H_\leftarrow = \{h_i^- \mid i \in \mathbb{N}\}$ and $H_\rightarrow = \{h_i^+ \mid i \in \mathbb{N}\}$ with $h_0^- > h_1^- > \dots$ and $h_0^+ < h_1^+ < \dots$.

- The pair $(H_\leftarrow, H_\rightarrow)$ is a *k-homogeneous factorisation of ξ into (u, v, w)* if
  - $\xi[i, j-1] \equiv_k u$ for all $i, j \in H_\leftarrow$ with $i < j$,
  - $\xi[i, j-1] \equiv_k v$ for all $i \in H_\leftarrow$ and $j \in H_\rightarrow$ with $i < j$ and
  - $\xi[i, j-1] \equiv_k w$ for all $i, j \in H_\rightarrow$ with $i < j$.
- The pair $(H_\leftarrow, H_\rightarrow)$ is *k-homogeneneous for ξ* if it is a k-homogeneous factorisation of ξ into some finite words (u, v, w).
- The pair $(H_\leftarrow, H_\rightarrow)$ is *uniformly homogeneous for ξ* if, for all $k \in \mathbb{N}$, the pair $(\{h_i^- \mid i \geq k\}, \{h_i^+ \mid i \geq k\})$ is k-homogeneous for ξ.

Let ξ be a bi-infinite word split into an ω^*-word $\xi_\leftarrow$ and an ω-word $\xi_\rightarrow$. As for any ω-word, there exists a uniformly homogeneous set $H_\rightarrow$ for $\xi_\rightarrow$. Symmetrically, there exists a set $H_\leftarrow \subseteq \widetilde{\mathbb{N}}$ that is "uniformly homogeneous" for $\xi_\leftarrow$. Then the pair $(H_\leftarrow, H_\rightarrow)$ is a uniformly homogeneous pair for $\xi = \xi_\leftarrow \xi_\rightarrow$.

▶ **Lemma 20.** *Let ξ be a recursive bi-infinite word with a decidable* MSO*-theory. Then the* MSO*-theories of $\xi_{\leftarrow} = \xi(-\infty, -1]$ and of $\xi_{\rightarrow} = \xi[0, \infty)$ are both decidable.*

Proof. We handle the cases of recurrent and non-recurrent words separately.

First let ξ be non-recurrent. Then some word $u \in F(\xi)$ has a leftmost or a rightmost occurrence, at some position $x \in \mathbb{Z}$ which is definable in FO. Hence, also the positions -1 and 0 are definable. Hence the MSO-theories of $\xi_{\leftarrow}$ and of $\xi_{\rightarrow}$ can be reduced to that of ξ and are therefore decidable.

Now let ξ be recurrent. By Thm. 14, ξ has a recursive indicator of recurrence $(\mathrm{rec}_{\leftarrow}, \mathrm{rec}_{\rightarrow})$. Define the functions $f, g\colon \mathsf{Sent} \to \mathbb{N} \cup \{\top\}$ as follows:

$$f(\varphi) = \begin{cases} \top & \text{if } \mathrm{rec}_{\leftarrow}(\varphi) = \top \\ 0 & \text{if } \mathrm{rec}_{\leftarrow}(\varphi) \geq 0 \\ |\mathrm{rec}_{\leftarrow}(\varphi)| - 1 & \text{otherwise} \end{cases} \quad \text{and} \quad g(\varphi) = \begin{cases} \top & \text{if } \mathrm{rec}_{\rightarrow}(\varphi) = \top \\ 0 & \text{if } \mathrm{rec}_{\rightarrow}(\varphi) < 0 \\ \mathrm{rec}_{\rightarrow}(\varphi) & \text{otherwise} \end{cases}$$

Exploiting the properties of $\mathrm{rec}_{\leftarrow}$ and $\mathrm{rec}_{\rightarrow}$, it is then routine to check that f, g are indicators of recurrences for the two ω-words $\xi_{\leftarrow}^R$ and $\xi_{\rightarrow}$. Note that $\xi_{\leftarrow}^R$ and $\xi_{\rightarrow}$ are recursive ω-words. Hence, by Thm. 5, the MSO-theories of $\xi_{\leftarrow}^R$ and of $\xi_{\rightarrow}$ are both decidable. ◀

▶ **Theorem 21.** *A recursive bi-infinite word ξ has a decidable* MSO*-theory if and only if there exists a recursive uniformly homogeneous pair for ξ.*

Proof. Suppose $\mathsf{MTh}(\xi)$ is decidable. By Lemma 20, the MSO-theories of $\xi_{\leftarrow}^R = \xi(-\infty, -1]^R$ and of $\xi_{\rightarrow} = \xi[0, \infty)$ are both decidable. Consequently, by Thm. 7, there are recursive uniformly homogeneous factorisations $H_{\leftarrow}^R, H_{\rightarrow} \subseteq \mathbb{N}$ for $\xi_{\leftarrow}^R$ and $\xi_{\rightarrow}$ into (x^R, y^R) and (y', z), respectively. Deleting, if necessary, the minimal element from $H_{\leftarrow}^R$, we can assume $0 \notin H_{\leftarrow}^R$. We set $H_{\leftarrow} = \{-n \mid n \in H_{\leftarrow}^R\} \subseteq \widetilde{\mathbb{N}}$ and show that $(H_{\leftarrow}, H_{\rightarrow})$ is a uniformly homogeneous pair for ξ: Let $H_{\leftarrow} = \{h_i^- \mid i \in \mathbb{N}\}$ and $H_{\rightarrow} = \{h_i^+ \mid i \in \mathbb{N}\}$ such that $h_0^- > h_1^- > \dots$ and $h_0^+ < h_1^+ < \dots$.

- Let $j > i \geq k$. Then $\xi[h_i^- + 1, h_j^-] = \xi_{\leftarrow}[h_i^- + 1, h_j^-] = (\xi_{\leftarrow}^R[-h_j^-, -h_i^- - 1])^R \equiv_k y^R$.
- Let $i, j \geq k$. Then $\xi[h_i^-, h_j^+ - 1] = \xi_{\leftarrow}[h_i^- + 1, 0]\, \xi_{\rightarrow}[0, h_j^+ - 1] \equiv_k xy'$
- Let $j > i \geq k$. Then $\xi[h_i^+, h_j^+ - 1] = \xi_{\rightarrow}[h_i^+, h_j^+ - 1] \equiv_k z$.

Hence the pair $(\{h_i^- \mid i \geq k\}, \{h_i^+ \mid i \geq k\})$ is a k-homogeneous factorisation of ξ into (y^R, xy', z). Since k is arbitrary, $(H_{\leftarrow}, H_{\rightarrow})$ is uniformly homogeneous for ξ. Since these two sets are clearly recursive, this proves the first implication.

Conversely, suppose there exists a recursive uniformly homogeneous pair $(H_{\leftarrow}, H_{\rightarrow})$ for ξ. Then the sets $H_{\leftarrow}^R = \{|n| \mid n \in H_{\leftarrow} \cap \widetilde{\mathbb{N}}\}$ and $H_{\rightarrow} \cap \mathbb{N}$ are recursive and uniformly homogeneous for $\xi_{\leftarrow}^R$ and $\xi_{\rightarrow}$, resp. Since $\xi_{\leftarrow}$ and $\xi_{\rightarrow}$ are both recursive, we can apply Thm. 7. Hence the infinite words $\xi_{\leftarrow}$ and $\xi_{\rightarrow}$ both have decidable MSO-theories. Since $\xi = \xi_{\leftarrow}\xi_{\rightarrow}$, the MSO-theory of ξ is decidable. ◀

We next show that we cannot hope to extend Thm. 21 to non-recursive words:

▶ **Theorem 22.** *There exists a recurrent r.e. bi-infinite word ξ with decidable* MSO*-theory such that there is no r.e. uniformly homogeneous pair for ξ.*

Proof. We prove this theorem by constructing a recurrent bi-infinite word ξ such that the set $F(\xi)$ of factors is $\{0,1\}^*$. Hence ξ has decidable MSO-theory by Thm. 14.

There is a computable function $f\colon \mathbb{N}^2 \to \mathbb{N}$ such that the following hold:

- $\Phi_{f(e,s)}$ is total and $W_{f(e,s)} \subseteq \{0, 1, \dots, s\}$ for any $e, s \in \mathbb{N}$.
- $W_e = \bigcup_{s \in \mathbb{N}} W_{f(e,s)}$ for any $e \in \mathbb{N}$.

In the following, we fix the function f and write $W_{e,s}$ for $W_{f(e,s)}$. Furthermore, we fix some recursive enumeration $u_0, u_1, \dots$ of the set $\{0,1\}^+$ of non-empty finite words.

5.2.1 Construction

By induction on $s \in \mathbb{N}$, we construct tuples

$$t_s = (w_s, m_{0,s}, m_{1,s}, \ldots, m_{s,s}, P_s) \in \{0,1\}^* \times \mathbb{N}^{s+1} \times 2^{\{0,\ldots,s\}} \qquad \text{such that}$$

- $m_{i,s} + |u_i| \leq m_{i+1,s}$ for all $0 \leq i < s$ and $m_{s,s} + |u_s| \leq |w_s|$ (in particular, $|w_s| > s$),
- $w_s[m_{i,s}, m_{i,s} + |u_i| - 1] = u_i$ for all $0 \leq i \leq s$, and
- for all $e \in P_s$, there exist $a, b \in W_e$ with $a < b < |w_s|$ and $w_s[a, b-1] \in 1^*$.

Set $w_0 = u_0$, $m_{0,0} = 0$, and $P_0 = \emptyset$. Then the inductive invariant holds for the tuple $t_0 = (w_0, m_0, P_0)$.

Now suppose the tuple t_s has been constructed. Let H_{s+1} denote the set of indices $0 \leq e \leq s+1$ with $e \notin P_s$ such that $W_{e,s}$ contains at least two numbers $a > b \geq m_{e,s}$. In the construction of the tuple t_{s+1}, we distinguish two cases:

- 1st case: $H_{s+1} = \emptyset$. Then set $w_{s+1} = w_s u_{s+1}$, $m_{i,s+1} = m_{i,s}$ for $0 \leq i \leq s$, $m_{s+1,s+1} = |w_s|$, and $P_{s+1} = P_s$. Since the inductive invariant holds for the tuple t_s, it also holds for the newly constructed tuple t_{s+1}.
- 2nd case: $H_{s+1} \neq \emptyset$. Let e_{s+1} be the minimal element of H_{s+1} and let a_{s+1} and b_{s+1} be the minimal elements of $W_{e_{s+1},s}$ satisfying $m_{e,s} \leq a_{s+1} < b_{s+1}$. Then set
  - $w_{s+1} = w_s[0, a_{s+1}-1]\, 1^{b_{s+1}-a_{s+1}}\, w_s[b_{s+1}, |w_s|-1]\, u_{e_{s+1}} u_{e_{s+1}+1} \ldots u_{s+1}$ (in other words, the words $u_{e_{s+1}}$ up to u_{s+1} are appended to w_s and the positions between a_{s+1} and $b_{s+1} - 1$ are set to 1).
  - $m_{i,s+1} = \begin{cases} m_{i,s} & \text{if } i < e_{s+1} \\ |w_s u_{e_{s+1}} u_{e_{s+1}+1} \ldots u_{i-1}| & \text{if } e_{s+1} \leq i \leq s+1 \end{cases}$
  - $P_{s+1} = P_s \cup \{e_{s+1}\}$

  The first two conditions of the inductive invariant are obvious. Regarding the last one, let $e \in P_{s+1}$. If $e \neq e_{s+1}$, then $e \in P_s$ and therefore there exist $a, b \in W_e$ with $a < b < |w_s| < |w_{s+1}|$ such that $w_s[a, b-1] \in 1^*$. Note that any position in w_s that carries 1 also carries 1 in w_{s+1}. Hence $w_{s+1}[a, b-1] \in 1^*$ as well. It remains to consider the case $e = e_{s+1}$. But then, by the very construction, $a_{s+1} < b_{s+1}$ belong to $W_{e_{s+1},s} \subseteq W_e$ and satisfy $w_{s+1}[a_{s+1}, b_{s+1} - 1] \in 1^*$.

This finishes the construction of the sequence of tuples t_s.

5.2.2 Verification

Let $\xi_\rightarrow$ be the ω-word with $\xi_\rightarrow(i) = 1$ iff there exists $s \in \mathbb{N}$ with $w_s(i) = 1$. Since the tuple t_{s+1} is computable from the tuple t_s, the word $\xi_\rightarrow$ is clearly recursively enumerable.

Furthermore, let $u \in \{0,1\}^+$. Then there exists $e \in \mathbb{N}$ with $u = u_e$. Note that $m_{e,s} \leq m_{e,s+1}$ for all $e, s \in \mathbb{N}$. Furthermore, $m_{e,s} < m_{e,s+1}$ iff $H_{s+1} \neq \emptyset$ and $e_{s+1} \leq e$. Since the numbers $e_{s'+1}$ for $s' \in \mathbb{N}$ (if defined) are mutually distinct, there exists $s \in \mathbb{N}$ such that $e_{t+1} > e$ and therefore $m_{e,s} = m_{e,t}$ for all $t \geq s$. Consequently, $\xi_\rightarrow[m_{e,s}, m_{e,s} + |u_e| - 1] = w_s[m_{e,s}, m_{e,s} + |u_e| - 1] = u_e = u$. This means that $F(\xi_\rightarrow) = \{0,1\}^*$. It follows that $\xi_\rightarrow$ is recurrent.

Claim 1. If W_e is infinite, then $e \in \bigcup_{s \in \mathbb{N}} P_s$.

Proof of Claim 1. By contradiction, suppose this is not the case. Let $e \in \mathbb{N}$ be minimal with W_e infinite and $e \notin \bigcup_{s \in \mathbb{N}} P_s$. Since W_e is infinite, we get $e \in H_{s+1}$ for almost all $s \in \mathbb{N}$. By minimality of e, there is $s \in \mathbb{N}$ with $e = \min H_{s+1}$. But then $e_{s+1} = e$ and $e \in P_{s+1}$. **q.e.d.**

Claim 2. No recursively enumerable set W is uniformly homogeneous for the ω-word $\xi_{\to}$.

Proof of Claim 2. Suppose W is recursively enumerable and uniformly homogeneous for $\xi_{\to}$. Then W is infinite and there exists $e \in \mathbb{N}$ with $W = W_e$. By claim 1, there exists $s \in \mathbb{N}$ with $e \in P_s$. Hence there are $a, b \in W_e$ with $w_s[a, b-1] \in 1^*$ and therefore $\xi_{\to}[a, b-1] = w_s[a, b-1]$. There are $d > c > b$ in W_e such that $\xi_{\to}[c, d-1] \notin 1^*$. But then $\xi_{\to}[a, b-1]$ and $\xi_{\to}[c, d-1]$ do not have the same 1-type. Hence the set W_e is not 1- and therefore not uniformly homogeneous for $\xi_{\to}$. **q.e.d.**

Finally, let $\xi_{\leftarrow}$ be the reversal of $\xi_{\to}$ and consider the bi-infinite word $\xi = \xi_{\leftarrow}\,\xi_{\to}$. By Thm. 14, $\mathsf{MTh}(\xi)$ is decidable since ξ is recurrent and contains every finite word as a factor. Finally, suppose $(H_{\leftarrow}, H_{\to})$ is uniformly homogeneous for ξ. Then $H_{\to} \cap \mathbb{N}$ is uniformly homogeneous for $\xi_{\to}$. By claim 2, this set cannot be recursively enumerable. Hence $(H_{\leftarrow}, H_{\to})$ is not recursively enumerable either. ◀

5.3 A characterisation à la Rabinovich-Thomas II

We next extend the 2nd characterisation by Rabinovich and Thomas to bi-infinite words. Differently from the 1st characterisation, this also covers non-recursive bi-infinite words.

▶ **Definition 23.** Let ξ be some bi-infinite word and $\mathrm{tp}\colon \mathbb{N} \to \{0,1\}^+ \times \{0,1\}^+ \times \{0,1\}^+$. The function tp is a *type-function for* ξ if, for all $k \in \mathbb{N}$, the bi-infinite word ξ has a k-homogeneous factorisation into $\mathrm{tp}(k)$.

▶ **Theorem 24.** *Let ξ be a bi-infinite word. Then $\mathsf{MTh}(\xi)$ is decidable if and only if ξ has a recursive type-function.*

Proof. First suppose that $\mathsf{MTh}(\xi)$ is decidable. We have to construct a recursive type-function $\mathrm{tp}\colon \mathbb{N} \to (\{0,1\}^+)^3$. To this aim, let $k \in \mathbb{N}$. Then one can compute a finite sequence $\varphi_1, \ldots, \varphi_n$ of MSO-sentences of quantifier-rank k such that, for all finite words u and v, we have $u \equiv_k v$ if and only if $\forall 1 \le i \le n\colon u \models \varphi_i \iff v \models \varphi_i$. For finite words u, v, and w, consider the following statement:

$$\begin{aligned}
\exists H_{\leftarrow}, H_{\to}\colon \quad & \forall y \exists x, z\colon (x < y < z \wedge H_{\leftarrow}(x) \wedge H_{\to}(z)) \\
\wedge \quad & \forall x, y\colon (x < y \wedge H_{\leftarrow}(x) \wedge H_{\leftarrow}(y) \to \xi[x, y-1] \equiv_k u) \\
\wedge \quad & \forall x, y\colon ((H_{\leftarrow}(x) \wedge H_{\to}(y) \wedge x < y \to \xi[x, y-1] \equiv_k v) \\
\wedge \quad & \forall x, y\colon (x < y \wedge H_{\to}(x) \wedge H_{\to}(y) \to \xi[x, y-1] \equiv_k w)
\end{aligned}$$

This statement holds for a bi-infinite word ξ iff ξ has a k-homogeneous factorisation into (u, v, w). Using $\varphi_1, \ldots, \varphi_n$, the statements $\xi[x, y-1] \equiv_k u$ etc. can be expressed as MSO-formulas with free variables x and y. Since $\mathsf{MTh}(\xi)$ is decidable, we can decide (given k, u, v, and w) whether ξ has a k-homogeneous factorisation into (u, v, w). Since some k-homogeneous factorisation always exist, this allows to compute, from k, a tuple $\mathrm{tp}(k)$ such that ξ has a k-homogeneous factorisation into $\mathrm{tp}(k)$; tp is the wanted type function.

Conversely suppose that tp is a recursive type-function for ξ. To show that $\mathsf{MTh}(\xi)$ is decidable, let $\varphi \in \mathsf{Sent}$ be any MSO-sentence. Let k denote the quantifier-rank of φ. First, compute $\mathrm{tp}(k) = (u, v, w)$. Then $\xi \models \varphi$ iff $u^{\omega^*} v w^\omega \models \varphi$ which is decidable since this bi-infinite word is ultimately periodic on the left and on the right. ◀

6 How complicated are bi-infinite words with decidable MSO-theories?

By Thm. 14, non-recurrent bi-infinite words with decidable MSO-theory are recursive. In this section, we will show in a strong sense that this does not hold for recurrent bi-infinite words: there are "arbitrarily complicated" bi-infinite words with decidable MSO-theories.

▶ **Definition 25.** Let $L \subseteq \{0,1\}^*$ be a language. A word $u \in L$ is *left-determined in* L if for any $k \in \mathbb{N}$ there is exactly one word $vu \in L$ with $|v| = k$. Similarly, u is *right-determined in* L if for any $k \in \mathbb{N}$ there is exactly one word $uv \in L$ with $|v| = k$. The word $u \in L$ is *determined in* L if it is both left- and right-determined.

Intuitively, a word $w \in L$ is left-determined (right-determined) in L if it can be extended on the left (right) in a unique way.

▶ **Lemma 26.** *Let ξ be a recurrent bi-infinite word. The following are equivalent:*
(1) *ξ is periodic*
(2) *$F(\xi)$ contains a determined word*
(3) *$F(\xi)$ contains a right-determined word*
(3') *$F(\xi)$ contains a left-determined word*

Proof. For (1)→(2), let $\xi = u^{\omega^*}u^\omega$ be a periodic word. Then u is determined in $F(\xi)$. The direction (2)→(3) is trivial by the very definition.

For (3)→(1), suppose u is a right-determined word in $F(\xi)$. Choose $i < j$ such that $\xi[i, i+|u|-1] = \xi[j, j+|u|-1] = u$ (such a pair $i < j$ exists since ξ is recurrent). With $p = j-i$, we claim $\xi(n) = \xi(n+p)$ for all $n \in \mathbb{Z}$: First let $n \geq j+|u|$. Then $\xi[i,n]$ and $\xi[j,n+p]$ are two words from $F(\xi)$ that both start with u. We have $|\xi[i,n]| = n-i-1 = n+p-j-1 = |\xi[j,n+p]|$. Since u is right-determined, this implies $\xi[i,n] = \xi[j,n+p]$ and therefore $\xi(n) = \xi(n+p)$. Consequently, $\xi[j+|u|,\infty) = \xi[j+|u|, j+|u|+p]^\omega$. Next let $n < j+|u|$. Since ξ is recurrent, there is $k < n$ with $\xi[k, k+|u|-1] = u$. Since u is right-determined, this implies $\xi[k,\infty) = \xi[j+|u|,\infty) = \xi[j+|u|, j+|u|+p]^\omega$ and therefore in particular $\xi(n) = \xi(n+p)$. The implications (2)→(3')→(1) are shown analogously. ◀

Lemma 26 states that a recurrent non-periodic bi-infinite word does not contain any left-determined or right-determined factor, and thus can be extended in both directions (left and right) in at least two ways. This observation allows to prove the following:

▶ **Lemma 27.** *Let ξ be a recurrent non-periodic bi-infinite word. For any set $A \subseteq \mathbb{N}$, there is a recurrent bi-infinite word ξ_A such that $F(\xi) = F(\xi_A)$, $(A, F(\xi)) \leq_T \xi_A$, and $\xi_A \leq_T (A, F(\xi))$.*

Proof. Let $w_0, w_1, \ldots$ be the enumeration of $F(\xi)$ in length-lexicographic order. Note that this is recursive in $F(\xi)$. There is also an effective enumeration of all pairs of words of the same length, say $(\ell_0, r_0), (\ell_1, r_1), \ldots$. Now let $A \subseteq \mathbb{N}$ be arbitrary. We will construct a sequence of tuples $t_s = (u_s, v_s, x_s, y_s) \in (\{0,1\}^*)^4$ such that, for all $s \in \mathbb{N}$, the finite word

$$\begin{aligned} z_s &= w_s y_s v_s \; z_{s-1} \; u_s x_s w_s \\ &= w_s y_s v_s \, w_{s-1} y_{s-1} v_{s-1} \ldots w_0 y_0 v_0 \; u_0 x_0 w_0 \ldots u_{s-1} x_{s-1} w_{s-1} \, u_s x_s w_s \end{aligned}$$

belongs to $F(\xi)$ (the bi-infinite word ξ_A will be the "limit" of these words).

To start with $s = 0$ note the following: since ξ is recurrent and $w_0 \in F(\xi)$, the bi-infinite word ξ contains a factor of the form $w_0 x w_0$. Set $y_0 = x$ and $u_0 = v_0 = x_0 = \varepsilon$.

For the induction step, assume that we constructed the tuple t_s and that z_s is a factor of ξ. Since ξ is recurrent but not periodic, the word z_s is not right-determined in $F(\xi)$ by Lemma 26. Hence there are two distinct finite words u and u' of the same length such that $z_s u, z_s u' \in F(\xi)$. For (u, u'), choose the first such pair in the effective enumeration $(\ell_i, r_i)_{i\in\mathbb{N}}$. If $s \in A$, then set $u_{s+1} = u$, otherwise set $u_{s+1} = u'$. Now the word $z_s u_{s+1}$ is a factor of ξ. Since ξ is recurrent, there is $x_{s+1} \in \{0,1\}^*$ such that $z_s u_{s+1} x_{s+1} w_{s+1}$ is a factor of ξ – choose x_{s+1} length-lexicographically minimal among all possible such words.

To choose v_{s+1} and y_{s+1}, we proceed symmetrically to the left: $z'_s = z_s u_{s+1} x_{s+1} w_{s+1}$ is a factor of ξ that is not left-determined. Hence there exists a pair of distinct words v and v' of the same length with $vz'_s, v'z'_s \in F(w)$. Choose this pair minimal in the effective enumeration. If $s \in A$, then set $v_{s+1} = v$, otherwise set $v_{s+1} = v'$. Now there is $y_{s+1} \in \{0,1\}^*$ with $w_{s+1} y_{s+1} v_{s+1} z'_s \in F(\xi)$ since ξ is recurrent. Choosing y_{s+1} length-lexicographically minimal completes the construction of the tuple t_{s+1} and therefore the inductive construction of all the tuples t_s. Now set $\xi_A = \cdots w_1 y_1 v_1\, w_0 y_0 v_0\, u_0 x_0 w_0\, u_1 x_1 w_1 \cdots$. Observe the following:

- If $u \in F(\xi)$, then there exists $s \in \mathbb{N}$ such that $u \in F(z_s)$. Hence $F(\xi) \subseteq F(\xi_A)$.
- Let $u \in F(\xi_A)$. There exists $s \in \mathbb{N}$ such that $u \in F(z_s)$. In particular, $F(\xi_A) \subseteq F(\xi)$. Since z_s is a factor of ξ, there are infinitely many $i \in \mathbb{N}$ such that z_s (and therefore u) is a factor of w_i. Hence the word ξ_A is recurrent.

Since the above describes how to compute the bi-infinite word ξ_A using the oracles A and $F(w)$, we get $\xi_A \leq_T (A, F(\xi))$.

It remains to be shown that $A \leq_T (\xi_A, F(\xi))$ holds: To determine whether $s \in A$ suppose we already know which of the natural numbers $i < s$ belong to A. Then the construction of ξ_A above allows to build t_s using the oracle $F(\xi)$. Now construct t_{s+1} assuming $s \in A$ again using the oracle $F(\xi)$. If the resulting word z_{s+1} is an initial segment of ξ_A, then $s \in A$. Otherwise, $s \notin A$. ◀

From this lemma and Thm. 14, we get immediately that indeed, every decidable theory of some recurrent bi-infinite word is represented in every Turing-degree:

▶ **Theorem 28.** *Let ξ be a recurrent non-periodic bi-infinite word and $\mathbf{a}$ a Turing-degree above the degree of $\mathsf{MTh}(\xi)$. Then $\mathbf{a}$ contains a bi-infinite word ξ_A with $\mathsf{MTh}(\xi_A) = \mathsf{MTh}(\xi)$.*

7 How many indistinguishable bi-infinite words are there?

If α and β are MSO-equivalent ω-words, then $\alpha = \beta$. In this final section we study this question for bi-infinite words. Shift-equivalence and period will be important notions in this context: two bi-infinite words ξ and ζ are *shift-equivalent* if there is $p \in \mathbb{N}$ with $\xi(n) = \zeta(n+p)$ for all $n \in \mathbb{Z}$. Furthermore, the period of the bi-infinite word ξ is the least natural number $p > 0$ with $\xi(n) = \xi(n+p)$ for all $n \in \mathbb{Z}$ – clearly, the period need not exist. To count the number of MSO-equivalent bi-infinite words, we need a characterisation when two bi-infinite words are MSO-equivalent.

▶ **Theorem 29** ([6, Chp. 9, Thm. 6.1])**.** *Two bi-infinite words ξ and ζ are MSO-equivalent if and only if one of the following conditions is satisfied:*

1. *ξ and ζ are shift-equivalent.*
2. *ξ and ζ are recurrent and have the same set of factors.*

This characterisation is the central ingredient in the proof of the following result:

▶ **Theorem 30.** *Let ξ be a bi-infinite word.*

(a) *If ξ is periodic, then the cardinality of the type of ξ is finite and equals the period of ξ.*

(b) *If ξ is non-recurrent, then the cardinality of the type of ξ is $\aleph_0$.*

(c) *If ξ is recurrent and non-periodic, then the cardinality of the type of ξ is $2^{\aleph_0}$.*

Proof.

(a) Let p be the period of ξ. Since p is minimal, there are precisely p distinct bi-infinite words that are shift-equivalent with ξ. Since shift-equivalent words are MSO-equivalent, the type of ξ contains at least p elements. It remains to be shown that no further MSO-equivalent word exists. So let ζ be some MSO-equivalent word. Then ζ is p-periodic since ξ (and therefore ζ) satisfies $\forall x\colon (P(x) \Leftrightarrow P(x+p))$ and does not satisfy $\forall x\colon (P(x) \Leftrightarrow P(x+q))$ for any $1 \leq q < p$. Furthermore $u = \xi[1,p]$ is a factor of ξ and therefore of ζ of length p. Hence $\zeta = u^{\omega^*}u^{\omega}$.

(b) This claim follows immediately from Thm. 29.

(c) This follows from Thm. 28 as there are $2^{\aleph_0}$ Turing-degree above any Turing-degree. ◀

References

1 Alexis Bés and Alexander Rabinovich. Decidable expansions of labelled linear orderings. *Logical Methods in Computer Science*, 7(2), 2011.

2 J. Richard Büchi. On a decision method in restricted second order arithmetic. In *Proceedings of the Interantional Congress on Methodology and Philosophy of Science*, pages 1–11. Stanford University Press, 1962.

3 J. Richard Büchi and Lawrence Landweber. Definability in the monadic second-order theory of successor. *Journal of Symbolic Logic*, 34:166–170, 1969.

4 Olivier Carton and Wolfgang Thomas. The monadic theory of morphic infinite words and generalizations. *Information and Computation*, 176(1):51–56, 2002.

5 Calvin Elgot and Michael O. Rabin. Decidability and undecidability of extensions of second (first) order theory of (generalized) successor. *J. of Symbolic Logic*, 31(2):169–181, 1996.

6 Dominique Perrin and Jean-Éric Pin. *Infinite words: automata, semigroups, logic and games*, volume 141 of *Pure and Applied Mathematics Series*. Elsevier, 2004.

7 Dominique Perrin and Paul Schupp. Automata on the integers, recurrence distinguishability, and the equivalence and decidability of monadic theories. In *Proceedings of the Symposium on Logic in Computer Science*, pages 301–304. IEEE Computer Society Press, 1986.

8 Alexander Rabinovich. On decidability of monadic logic of order over the naturals extended by monadic predicates. *Information and Computation*, 205(6):870–889, 2007.

9 Alexander Rabinovich and Wolfgang Thomas. Decidable theories of the ordering of natural numbers with unary predicates. In *Computer Science Logic, 20th International Workshop, CSL 2006, 15th Annual Conference of the EACSL, Szeged, Hungary, September 25-29, 2006, Proceedings*, Lecture Notes in Computer Science, pages 562–574. Springer, 2006.

10 Alexei Semenov. Decidability of monadic theories. In *Mathematical Foundations of Computer Science 1984, Praha, Czechoslovakia, September 3-7, 1984, Proceedings*, Lecture Notes in Computer Science, pages 162–175. Springer, 1984.

11 Alexei Semenov. Logical theories of one-place functions on the set of natural numbers. *Mathematics of the USSR – Izvestia*, 22:587–618, 1984.

12 Saharon Shelah. The monadic theory of order. *Annals of Mathematics*, 102:379–419, 1957.

13 Dirk Siefkes. Decidable extensions of monadic second order successor arithmetic. *Automatentheorie und formale Sprachen*, pages 441–472, 1969.

14 Robert Soare. *Recursively enumerable sets and degrees: A Study of Computable Functions and Computably Generated Sets*. Perspectives in Mathematical Logic. Springer, 1987.

15 Wolfgang Thomas. Languages, automata, and logic. In *Handbook of Formal Languages*, volume 3, pages 389–455. Springer, 1997.

A Coalgebraic Decision Procedure for WS1S

Dmitriy Traytel

Fakultät für Informatik, Technische Universität München, Germany
traytel@in.tum.de

Abstract

Weak monadic second-order logic of one successor (WS1S) is a simple and natural formalism to specify regular properties. WS1S is decidable, although the decision procedure's complexity is non-elementary. Typically, decision procedures for WS1S exploit the logic–automaton connection, i.e., they escape the simple and natural formalism by translating formulas into equally expressive regular structures such as finite automata, regular expressions, or games. In this work, we devise a coalgebraic decision procedure for WS1S that stays within the logical world by directly operating on formulas. The key operation is the derivative of a formula, modeled after Brzozowski's derivatives of regular expressions. The presented decision procedure has been formalized and proved correct in the interactive proof assistant Isabelle.

1998 ACM Subject Classification F.4.3 Formal Languages

Keywords and phrases WS1S, decision procedure, coalgebra, Brzozowski derivatives, Isabelle

Digital Object Identifier 10.4230/LIPIcs.CSL.2015.487

1 Introduction

In his seminal work [8], Büchi envisioned weak monadic second-order logic of one successor (WS1S) to become a "more conventional formalism [that] can be used in place of regular expressions [...] for formalizing conditions on the behavior of automata". This vision became truth – WS1S has been used to encode decision problems in hardware verification [3], program verification [22], network verification [4], synthesis [19], as well as many others.

WS1S is a logic that supports first-order quantification over natural numbers and second-order quantification over finite (therefore "Weak") sets of natural numbers, and beyond this has few additional special predicates, such as $<$ to compare first-order variables. Equivalence of WS1S formulas is decidable, although the complexity for deciding it is non-elementary [27]. Nevertheless, the MONA tool [20] shows that the daunting theoretical complexity can often be overcome in practice by employing a multitude of smart optimizations. Similarly to Büchi, MONA's manual [24] calls WS1S a "simple and natural notation" for regular languages.

Traditionally[1], decision procedures for WS1S do not try to benefit from the conventional, simple, and natural logical notation. Instead, by exploiting the logic–automaton connection, formulas are translated into finite automata which are then minimized. During the translation all the rich algebraic formula structure including binders and high-level constructs is lost. On the other hand, the subsequent minimization might have benefited from some simplifications on the formula level.

Concerning the algebraic structure, regular expressions are situated somewhere in between WS1S formulas and automata. In earlier work [40, 39], we propose a semantics-preserving translation of WS1S formulas into regular expressions. Thereby, equivalence of formulas is

[1] The only notable exception, we are aware of, is the decision procedure implemented in the Toss tool [17] (Sect. 7).

24th EACSL Annual Conference on Computer Science Logic (CSL 2015).
Editor: Stephan Kreutzer; pp. 487–503

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

reduced to equivalence of regular expressions. To decide the latter, we employ a coalgebraic procedure based on an ε-acceptance test and Brzozowski derivatives [7] – the coalgebra structure on regular expressions [32].

In this paper, we go one step further by defining a syntactic coalgebra structure (a short recapitulation of language coalgebra is given Sect. 2) directly on WS1S formulas (whose syntax and semantics are introduced in Sect. 3). The main contributions are:

- We define a symbolic *derivative* operation for a WS1S formula (Sect. 4).
- We define an *acceptance test* whether a formula holds in the empty interpretation (Sect. 5).
- Taking the two above notions together, we obtain a decision procedure for WS1S that operates only on formulas (Sect. 6).
- We formalize the procedure in Isabelle/HOL [29] and prove its correctness.

On the one hand, the obtained decision procedure can be considered an elegant toy – implementable only with a few hundred lines of Standard ML (and thus well-suited for formalization) and teachable in class. At this stage, our intention is not to compete with MONA's thousands of lines of tricky performance optimizations (which still can be outperformed by our procedure on well selected formulas), but rather with another verified procedure from our own previous work [40, 39]. Here, the procedure presented in this work shows a significant performance improvement (Sect. 6).

On the other hand, being able to safely decide small formulas is already of some value. In an experiment, we have randomly generated small formulas and compared the outcome of our verified algorithm with the results produced by MONA and two other (less mature) tools: Toss [17] and dWiNA [14]. As the result, we were able to point the developers of the latter two tools to corner cases where their tools gave a wrong answer [36, 35]. Admittedly, one could have performed the same test without a formalized decision procedure, but then in case of a discrepancy, determining who is right might be difficult with random formulas.

Finally, we are confident that symbolic decision procedures need not to hide behind traditional automata-based ones in terms of performance in general. Sect. 7 on related work supports this claim by several successful examples, which we expect to carry over to our setting.

Notational Conventions. We employ a mixture of standard mathematical and functional programming notations. Function definitions are written in pseudo-Standard ML and we indicate the ML types, written postfix, where we consider them helpful. We assume no familiarity with Isabelle/HOL. Formal languages are sets of words. Words are represented by α *list*, finite lists of elements of type α. Lists are either empty ($[]$) or constructed by the infix operator $::$ of type $\alpha \Rightarrow \alpha\ list \Rightarrow \alpha\ list$. The n-th element of the list xs is accessed by $xs[n]$ (zero-based). The length of xs is written $|xs|$; applied to a set this notation also denotes the set's cardinality. Lists can be iterated over using the standard combinator $\mathsf{fold} : (\alpha \Rightarrow \beta \Rightarrow \beta) \Rightarrow \alpha\ list \Rightarrow \beta \Rightarrow \beta$ with the characteristic equations $\mathsf{fold}\ f\ []\ b = b$ and $\mathsf{fold}\ f\ (x :: xs)\ b = f\ x\ (\mathsf{fold}\ f\ xs\ b)$. The type *bool* is inhabited by two elements: $\mathsf{1}$ and $\mathsf{0}$.

2 Languages Are Coalgebras

The coalgebraic view on formal languages is, to our knowledge, due to Rutten [32]. The key observation is that the final coalgebra $(\alpha\ lang, \mathsf{out} : \alpha\ lang \Rightarrow F(\alpha\ lang))$ of the functor $F(S) = bool \times (\alpha \Rightarrow S)$ exists and is isomorphic to the standard set of words view on languages.

Using the finality of α *lang* we can define functions of type $\tau \Rightarrow \alpha\ lang$ by providing an F-coalgebra on τ, i.e., a function of type $\tau \Rightarrow bool \times (\alpha \Rightarrow \tau)$ that essentially describes

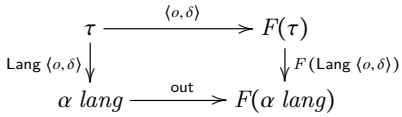


Figure 1 Unique morphism Lang $\langle o, \delta\rangle$ to the final coalgebra $(\alpha\ lang, \mathsf{out})$.

a deterministic (not necessarily finite) automaton without an initial state. To clarify this automaton analogy, it is customary to present the F-coalgebra as two functions $\langle o, \delta\rangle$ with τ being the states of the automaton, $o : \tau \Rightarrow bool$ denoting accepting states, and $\delta : \alpha \Rightarrow \tau \Rightarrow \tau$ being the transition function. From a given $\langle o, \delta\rangle$, we obtain the function $\mathsf{Lang}\ \langle o, \delta\rangle : \tau \Rightarrow \alpha\ lang$ that assigns to a separately given initial state $t : \tau$ the language $\mathsf{Lang}\ \langle o, \delta\rangle\ t : \alpha\ lang$ and makes the diagram in Figure 1 commute.

Exploiting the isomorphism to the set of words representation, the final coalgebra $\alpha\ lang$ itself corresponds to the automaton, whose states are languages, acceptance is given by $o\ L = [\,] \in L$, and the transition function by left quotients $\delta\ a\ L = (L)_a = \{w \mid aw \in L\}$. Note that $\mathsf{Lang}\ \langle o, \delta\rangle\ (L : \alpha\ lang) = L$ in this case.

A second consequence of the finality of $\alpha\ lang$ is the coinduction principle. A relation $R : \alpha\ lang \Rightarrow \alpha\ lang \Rightarrow bool$ is a *bisimulation* iff for all languages $L, K : \alpha\ lang$ such that $R\ L\ K$ holds, we have that $[\,] \in L \Leftrightarrow [\,] \in K$ and for all $a : \alpha$ it holds $R\ (L)_a\ (K)_a$. We write $L \sim K$ if there exists a bisimulation R such that $R\ L\ K$ and call such L and K *bisimilar*.

▶ **Theorem 1** (Coinduction). *Assume $L \sim K$. Then $L = K$.*

A necessary prerequisite for deciding bisimilarity of languages is that the languages in question admit a finite syntactic representation.[2] One example of such a finite syntactic representation are regular expressions. Although later we will work with a different finite syntactic representation (WS1S formulas), it is nevertheless instructive to shortly outline a coalgebraic decision procedure for regular expression equivalence.

Regular expressions are defined inductively as

$$RE = \varnothing \mid \boldsymbol{\varepsilon} \mid a \mid RE + RE \mid RE \cdot RE \mid RE^*$$

where $a \in \Sigma$ for a fixed alphabet $\Sigma : \alpha\ list$. The language of a regular expression r is defined as $\mathsf{L}\ r = \mathsf{Lang}\ \langle \varepsilon, \mathsf{d}\rangle\ r$ where $\varepsilon : RE \Rightarrow bool$ is the test whether the regular expression accepts the empty word and $\mathsf{d} : \alpha \Rightarrow RE \Rightarrow RE$ is the Brzozowski derivative [7] defined as usual:

$$\begin{array}{llll}
\varepsilon\ \varnothing & = 0 & \mathsf{d}\ a\ \varnothing & = \varnothing \\
\varepsilon\ \boldsymbol{\varepsilon} & = 1 & \mathsf{d}\ a\ \boldsymbol{\varepsilon} & = \varnothing \\
\varepsilon\ a & = 0 & \mathsf{d}\ a\ b & = \text{if } a = b \text{ then } \boldsymbol{\varepsilon} \text{ else } \varnothing \\
\varepsilon\ (r + s) & = \varepsilon\ r \vee \varepsilon\ s & \mathsf{d}\ a\ (r + s) & = \mathsf{d}\ a\ r + \mathsf{d}\ a\ s \\
\varepsilon\ (r \cdot s) & = \varepsilon\ r \wedge \varepsilon\ s & \mathsf{d}\ a\ (r \cdot s) & = \mathsf{d}\ a\ r \cdot s + \text{if } \varepsilon\ r \text{ then } \mathsf{d}\ a\ s \text{ else } \varnothing \\
\varepsilon\ (r^*) & = 1 & \mathsf{d}\ a\ (r^*) & = \mathsf{d}\ a\ r \cdot (r^*)
\end{array}$$

Note, that we exploit the coalgebraic view on regular expressions to define their semantics [21]. With the given definitions of the syntactic F-coalgebra on RE consisting of $\boldsymbol{\varepsilon}$ and d, the unique morphism L to the final coalgebra $\alpha\ lang$ corresponds to the language notion in the

[2] This is the case for regular languages, but also for e.g., context-free languages, where equivalence (and bisimilarity) is undecidable. I.e., the finite syntactic representations prerequisite is not sufficient.

set of words world (usually defined by recursion on RE). Moreover, for the unique morphism we obtain the expected characteristic theorems: $\varepsilon\, r \Leftrightarrow [\,] \in \mathsf{L}\, r$ and $\mathsf{L}(\mathsf{d}\, a\, r) = (\mathsf{L}\, r)_a$.

The coalgebraic decision procedure for regular expression equivalence iteratively constructs a relation $P : RE \Rightarrow RE \Rightarrow bool$ whose direct image under L is a bisimulation. For example, suppose we want to prove the regular expressions r and s being equivalent. We start with the pair (r, s), check $\varepsilon\, r \Leftrightarrow \varepsilon\, s$ and add it to the relation P. Then we close P under the pairwise derivative d for all letters of the alphabet. Whenever a new pair (t, u) is added to P, it is checked to fulfill $\varepsilon\, t \Leftrightarrow \varepsilon\, u$. Once P is closed under derivatives and all checks have been passed we know that it is a (syntactic) bisimulation and therefore by coinduction the languages of the input regular expressions r and s coincide.

The implementation bisim of this procedure employs a worklist algorithm to saturate P. In order to be reusable, bisim generalizes over d (abstract parameter δ), ε (abstract parameter o), syntactic representations of regular languages (types σ and τ). The abstract parameter ι is used as some initial transformation (or translation) of the syntactic representations. It will be of importance later when we instantiate σ with WS1S formulas.

```
bisim : α list ⇒ (σ ⇒ τ) ⇒ (α ⇒ τ ⇒ τ) ⇒ (τ ⇒ bool) ⇒ σ ⇒ σ ⇒ bool
bisim Σ ι δ o s t =
  let
    closure ([], _) = 1
    closure ((s, t) :: ws, P) = if o s ≠ o t then 0 else
      let
        add_new a (ws, P) =
          let st = (δ s a, δ t a) in if st ∈ P then (ws, P) else (st :: ws, {st} ∪ P)
      in closure (fold add_new Σ (ws, P))
    st₀ = (ι s, ι t)
  in closure ([st₀], {st₀})
```

The presented algorithm is a slight generalization of a framework for deciding regular expression equivalence [30]. If it terminates, bisim decides language equivalence of σ (given notions of languages $\hat{L}$ and L for σ and τ respectively, and executable arguments ι, δ, o that behave well with respect to the language notions).

▶ **Theorem 2.** *Fix*

$$\Sigma : \alpha\ list, \quad L : \tau \Rightarrow \alpha\ lang, \quad \hat{L} : \sigma \Rightarrow \alpha\ lang, \quad \iota : \sigma \Rightarrow \tau, \quad \delta : \alpha \Rightarrow \tau \Rightarrow \tau, \text{ and } \quad o : \tau \Rightarrow bool.$$

Assume that for all $s : \sigma$, $t : \tau$, and $a \in \Sigma^*$ we have

$$L\, t \subseteq \Sigma^*, \quad L\,(\iota\, s) = \hat{L}\, s, \quad L\,(\delta\, a\, t) = (L\, t)_a, \quad o\, t \Leftrightarrow [\,] \in L\, t, \text{ and } \quad |\{\mathsf{fold}\ \delta\ w\ t \mid w \in \Sigma^*\}| < \infty.$$

Then $\mathsf{bisim}\ \Sigma\ \iota\ \delta\ o\ s\ s' \Leftrightarrow \hat{L}\, s = \hat{L}\, s'$.

Proof. Because of the finiteness assumption closure (hence also bisim) does always terminate (the set of pairs to be explored is finite).

$$\mathsf{bisim}\ \Sigma\ \iota\ \delta\ o\ s\ s' \overset{1}{\Leftrightarrow} L\,(\iota\, s) \sim L\,(\iota\, s') \Leftrightarrow \hat{L}\, s \sim \hat{L}\, s' \overset{2}{\Leftrightarrow} \hat{L}\, s = \hat{L}\, s'$$

Equivalence 1 is justified by the fact that the only possibility for closure to return 1 is to make the worklist ws empty. Whenever the worklist becomes empty, the set of processed pairs P is a bisimulation. The nontrivial direction of 2 is justified by coinduction. ◀

The instantiation $\mathsf{eqv}_{RE}\ \Sigma = \mathsf{bisim}\ \Sigma\ (\lambda r.\ r)\ (\lambda a\ r.\ |\mathsf{d}\ a\ r|_{\mathsf{ACI}})\ \varepsilon$, where $|-|_{\mathsf{ACI}}$ is a function computing some normal form of regular expressions with respect to associativity, commutativity, and idempotence of the + constructor, decides regular expression equivalence. This follows from Theorem 2 with $L = \hat{L} = \mathsf{Lang}\ \langle \varepsilon, \mathsf{d} \rangle$, where the finiteness assumption holds by a classic result due to Brzozowski [7].

3 Syntax and Semantics of WS1S

We briefly introduce WS1S, as well as a standard encoding of interpretations as formal words. More thorough introductions are given elsewhere [34, 24]. Formulas are defined inductively as

$$\Phi = \mathsf{T} \mid \mathsf{F} \mid \mathsf{FO}\ var \mid var < var \mid var \in var \mid \Phi \vee \Phi \mid \neg\Phi \mid \exists_1\ \Phi \mid \exists_2\ \Phi$$

There are two kinds of quantifiers: $\exists_1$ quantifies over *first-order* variables, which denote natural numbers; $\exists_2$ quantifies over *second-order* variables, which denote finite sets of natural numbers. We use *de Bruijn indices* for variable bindings – therefore, no names are attached to quantifiers. An occurrence of a bound variable is just an index of type $var = nat$, that refers to its binder by counting the number of quantifiers between the occurrence and the binder. For example, the formula $\exists_1\ \mathsf{FO}\ 0 \vee (\exists_2\ 1 \in 0)$ would be customarily written as $\exists_1 x.\ \mathsf{FO}\ x \vee (\exists_2 X.\ x \in X)$ indicating second-order variables by capital letters. The first 0 and the 1 refer to the outer first-order quantifier, while the second 0 refers to the inner second-order quantifier. *Loose* de Bruijn indices, that are lacking a binder, are considered to be free. For example, $0 < 1$ corresponds to the formula $x < y$ for free x and y.

De Bruijn indices must be manipulated with great care and are hard to read. However, we prefer their usage for three reasons: First, they enable equality of formulas to hold modulo renaming of variables without further ado. Second, for any formula, its free variables are naturally ordered by the de Bruijn index. On several occasions, we will benefit from this order by using a simple list, the n-th element of which is associated to the variable with the index n. A priori this does not seem to be an advantage over using a functional assignment. However one of the occasions will be the underlying alphabet for the formula's language – an alphabet consisting of arbitrary functions is not a good idea for a decision procedure. Third, de Bruijn indices were successfully employed in the formalization.

The usual abbreviations define conjunction $\varphi \wedge \psi = \neg(\neg\varphi \vee \neg\psi)$ and universal quantification $\forall_i\ \varphi = \neg\exists_i\ (\neg\varphi)$ for $i \in \{1, 2\}$.

The truth of a formula depends on an *interpretation* of its free variables. An interpretation assigns each free variable of a formula a value. In our case, an interpretation is represented by a list of finite sets of natural numbers. The n-th entry of an interpretation is the assignment to the free variable of the formula denoted by the loose index n. Therefore, a first-order variable x is at first also assigned a finite set $I[x]$, with the condition that it must be non-empty. The value assigned by the interpretation to x is then taken to be the minimum of $I[x]$ written, $\mathsf{Min}\ (I[x])$. The idea to encode first-order variables as non-empty sets where the minimum counts, rather than singleton sets comes from MONA [24, Sect. 3.2] and leads to a more efficient decision procedure (also in our setting).

We define the WS1S-semantics of a formula φ w.r.t. an interpretation I, written $I \vDash \varphi$.

$$
\begin{aligned}
I \vDash \mathsf{T} &= 1 \\
I \vDash \mathsf{F} &= 0 \\
I \vDash (x < y) &= I[x] \neq \{\} \wedge I[y] \neq \{\} \wedge \mathsf{Min}\,(I[x]) < \mathsf{Min}\,(I[y]) \\
I \vDash (x \in X) &= I[x] \neq \{\} \wedge \mathsf{Min}\,(I[x]) \in I[X] \\
I \vDash (\mathsf{FO}\ x) &= I[x] \neq \{\} \\
I \vDash (\varphi \vee \psi) &= I \vDash \varphi \vee I \vDash \psi \\
I \vDash (\neg\varphi) &= \neg(I \vDash \varphi) \\
I \vDash (\exists_1\, \varphi) &= \exists p.\ (\{p\} :: I) \vDash \varphi \\
I \vDash (\exists_2\, \varphi) &= \exists P.\ |P| < \infty \wedge (P :: I) \vDash \varphi
\end{aligned}
$$

Intuitively, T is truth, F is falsity, $x < y$ compares assignments of first-order variables, $x \in X$ checks that the assignment to x is contained in the assignment to X. The (less standard) formula $\mathsf{FO}\ x$ asserts that x is a first-order variable. The Boolean connectives and quantifiers behave as expected. De Bruijn indices allow us to conveniently extend the interpretation by simply prepending the value for the most recently quantified variable when recursively entering the scope of a quantifier.

If $I \vDash \varphi$ holds, we say I *satisfies* φ or I is a *model* of φ. Formulas and interpretations must be *wellformed*: e.g., first-order variables shall not be used as second-order variables, and vice versa; no out-of-bounds accesses to the interpretation happen in $\vDash$. To ease the presentation, we omit the formal definition of wellformedness (it can be found in our formalization [38]) and consider only wellformed formulas and interpretations.

Another aspect of $\vDash$, that we want to factor out at first is the native support for first-order quantification. Therefore, in Sects. 4 and 5, we will use a different semantics that treats both quantifiers alike, written $I \Vdash \varphi$. It only differs from $\vDash$ in the quantifier cases:

$$I \Vdash (\exists_i\, \varphi) = \exists P.\ |P| < \infty \wedge (P :: I) \Vdash \varphi \quad \text{for } i \in \{1, 2\}$$

In Sect. 6, we will see how a formula under the $\vDash$-semantics can be easily syntactically translated into an equivalent formula under the $\Vdash$-semantics by inserting $\mathsf{FO}\ x$ in the right places.

There exists an alternative semantics for monadic-second order logic on finite words: M2L(Str) [2, 23] (we drop the (Str) from now on). Early versions of MONA implemented the M2L semantics. While WS1S is considered to be number theoretic (as intended by Büchi), the M2L semantics fits more naturally into the realm of automata. The difference between the two semantics lies only in the treatment of quantifiers. In WS1S, a quantified variable may be assigned arbitrarily large sets. In M2L, allowed assignments are bounded by the some number $\#I$ greater than all numbers occurring in the interpretation I, i.e., $\forall x \in \bigcup_{i=0}^{|I|} I[i].\ x < \#I$. The number $\#I$ is intrinsic to an interpretation (although here for simplicity we pretend it is a separate value). For WS1S its choice does not matter for satisfiability.

Our goal is to develop a decision procedure for WS1S. However, a decision procedure for M2L will arise on the way as a natural by-product. Therefore, we introduce its semantics formally, written $I \vDash_< \varphi$. Again, only the quantifier cases differ from the definition of $\vDash$.

$$
\begin{aligned}
I \vDash_< (\exists_1\, \varphi) &= \exists p.\ (p < \#I) \wedge (\{p\} :: I) \vDash_< \varphi \\
I \vDash_< (\exists_2\, \varphi) &= \exists P.\ (\forall p \in P.\ p < \#I) \wedge (P :: I) \vDash_< \varphi
\end{aligned}
$$

Although equally expressive as WS1S and somehow unusual in its treatment of quantifiers, the logic M2L has certain advantages, e.g., it yields a better complexity for the bounded

model construction problem [2]. Therefore, M2L is not just an ad hoc intermediate construct, and obtaining a decision procedure for it (basically for free) is of some value.

As for WS1S, we first use a simplified version of M2L that ignores the quantifier types.

$$I \Vdash_< (\exists_i\, \varphi) = \exists P.\ (\forall p \in P.\ p < \#I) \wedge (P :: I) \Vdash_< \varphi \quad \text{for } i \in \{1, 2\}$$

Two formulas are equivalent if they have the same models. Thus, we consider the language of a formula to be the set of its models encoded as words. The encoding of models (or interpretations in general) is standard: a finite set of natural numbers X is transformed into a list of Booleans, where 1 in the n-th positions means that $n \in X$. For interpretations the injective encoding function enc applies this transformation pointwise and pads the lists with 0s at the end to have length $\#I$. For example for $I = [\{1, 2, 3\}, \{0, 2\}]$ with $\#I = 4$, we obtain $\mathsf{enc}\ I = [[0, 1, 1, 1], [1, 0, 1, 0]]$, which we transpose to obtain the formal word w_I over the alphabet $bool^{|I|}$ (each letter is a vector, i.e., a list of fixed length).

$$w_I = \left[\begin{pmatrix}0\\1\end{pmatrix}, \begin{pmatrix}1\\0\end{pmatrix}, \begin{pmatrix}1\\1\end{pmatrix}, \begin{pmatrix}1\\0\end{pmatrix}\right]$$

In w_I a row corresponds to an assignment to a variable. For first-order variables a row must contain at least one 1 and the first 1 from the left counts as the assigned value. Finally, the *(simplified) WS1S-language* of a formula is defined as $\mathsf{L}\ \varphi = \{w_I \mid I \Vdash \varphi\}$ and the *(simplified) M2L-language* as $\mathsf{L}_<\ \varphi = \{w_I \mid I \Vdash_< \varphi\}$. The real languages, that we are after, are defined in the same way but using $\vDash$ and $\vDash_<$ instead of $\Vdash$ and $\Vdash_<$, with some further wellformedness conditions on I and φ (Sect. 6).

4 Formula Derivatives

Brzozowski derivatives compute symbolically for a regular expression r the regular expression $\mathsf{d}\ a\ r$ whose language is the left quotient of r's language by a. A *formula derivative* is the analogous operation on a formula. Given a formula φ, its derivative $\delta\ v\ \varphi$ is a formula whose language is the left quotient of φ's language by the letter $v : bool^n$, where n is the number of free variables of φ. Formula derivatives are defined by primitive recursion as follows.

$$\begin{aligned}
\delta\ v\ \mathsf{T} &= \mathsf{T} \\
\delta\ v\ \mathsf{F} &= \mathsf{F} \\
\delta\ v\ (x < y) &= \begin{cases} x < y & \text{if } \neg v[x] \wedge \neg v[y] \\ \mathsf{FO}\ y & \text{if } v[x] \wedge \neg v[y] \\ \mathsf{F} & \text{otherwise} \end{cases} \\
\delta\ v\ (\varphi \vee \psi) &= \delta\ v\ \varphi \vee \delta\ v\ \psi \\
\delta\ v\ (\neg\varphi) &= \neg\delta\ v\ \varphi \\
\delta\ v\ (\exists_i\, \varphi) &= \exists_i\, (\delta\ (1 :: v)\ \varphi \vee \delta\ (0 :: v)\ \varphi) \quad \text{for } i \in \{1, 2\}
\end{aligned}$$

$$\delta\ v\ (\mathsf{FO}\ x) = \begin{cases} \mathsf{T} & \text{if } v[x] \\ \mathsf{FO}\ x & \text{otherwise} \end{cases}$$

$$\delta\ v\ (x \in X) = \begin{cases} x \in X & \text{if } \neg v[x] \\ \mathsf{T} & \text{if } v[x] \wedge v[X] \\ \mathsf{F} & \text{otherwise} \end{cases}$$

Here we see the first important usage of the $\mathsf{FO}\ x$: to establish the closure of Φ under δ.

Let us try to intuitively understand some of the base cases of the definition. The language of F is the empty set. The quotient of the empty set is again empty. Thus, the derivative of F should be again F. For $\mathsf{FO}\ x$, the language will contain all words which have a 1 in the x-th row in some letter. Thus, if we quotient this language by a letter that has a 0 in the x-th row, we should obtain the same language represented by formula $\mathsf{FO}\ x$. However, once the first 1 in the x-th row has been discovered by quotienting by a letter that has a 1 in the

x-th row, no further restrictions on the remaining words to be accepted remain – i.e., the quotient is the universal language represented by formula T.

Other base cases follow similarly. The most interesting recursive case is that of the existential quantifiers. The first observation that should be made is that if φ is a formula with $n+1$ free variables, then $\exists_i\,\varphi$ is a formula with n free variables (assuming that the bound variable 0 is used in φ). This implies that when we derive $\exists_i\,\varphi$ by v, we cannot derive φ by v, but only by some vector $b :: v$ because all variables (de Bruijn indices) are shifted. Since the Boolean b is unknown, we consider both possibilities: $b = 1$ and $b = 0$. The existential quantifier requires just one witness assignment for the quantified variable. This means only one of the possibilities has to hold – that is why the recursive calls $\delta\ (1 :: v)\ \varphi$ and $\delta\ (0 :: v)\ \varphi$ are connected by disjunction.

To reason more directly about interpretations and derivatives, we lift the list constructor $::$ on words to interpretations. This yields the operator CONS, with the characteristic property $w_{(\mathsf{CONS}\ v\ I)} = v :: w_I$ assuming $|v| = |I|$, defined as follows.

$$\begin{aligned}
&\mathsf{CONS}\ [\,]\ [\,] = [\,]\\
&\mathsf{CONS}\ (1 :: v)\ (X :: I) = (\{0\} \cup (X+1)) :: \mathsf{CONS}\ v\ I\\
&\mathsf{CONS}\ (0 :: v)\ (X :: I) = (X+1) :: \mathsf{CONS}\ v\ I
\end{aligned}$$

The set $X+1$ is the pointwise increment of X, namely $\{x+1 \mid x \in X\}$. Additionally, we require $\#(\mathsf{CONS}\ v\ I) = \#I + 1$. We obtain the following key characterization of the derivative (again modulo wellformedness considerations of I, φ, and v).

▶ **Theorem 3.** *$I \Vdash \delta\ v\ \varphi \Leftrightarrow \mathsf{CONS}\ v\ I \Vdash \varphi$ and $I \Vdash_< \delta\ v\ \varphi \Leftrightarrow \mathsf{CONS}\ v\ I \Vdash_< \varphi$*

Proof. Two straightforward inductions on φ. ◀

The set of all word derivatives $\{\mathsf{fold}\ \delta\ w\ \varphi \mid w \in \Sigma^*\}$ over some alphabet Σ is infinite. However, the syntactic identification of formulas that can be rewritten into each other using only associativity, commutativity and idempotence of $\vee$ (ACI) results in a finite search space which the decision procedure has to explore. This fundamental theorem mimics Brzozowski's result about dissimilar derivatives of regular expressions modulo ACI of + [7].

▶ **Theorem 4.** *A formula φ has finitely many distinct word derivatives modulo ACI of $\vee$.*

Proof. By induction on φ. The most interesting case is $\exists_i\,\varphi$. By IH we know that φ has a finite set D of distinct derivatives modulo ACI. Some of the formulas in D can be disjunctions. If we repeatedly split outermost disjunctions in D until none are left, we obtain a finite set X of disjuncts. Each word derivative $\mathsf{fold}\ \delta\ w\ (\exists_i\,\varphi)$ is ACI equivalent to some $\exists_i\ (\bigvee Y)$ for some $Y \subseteq X$. Since X is finite, its powerset is also finite. Hence, there are finitely many distinct $\mathsf{fold}\ \delta\ w\ (\exists_i\,\varphi)$ modulo ACI.

Other cases are either obvious or carry over smoothly from Brzozowski's proof. ◀

Note that Theorem 4 is purely syntactic: no semantic (or language) equivalence is involved. In particular, this theorem does not follow from the Myhill-Nerode theorem nor from the fact that there are only finitely many semantically inequivalent formulas of bounded quantifier rank. Although not all language equivalent formulas are also ACI equivalent, ACI suffices for gaining finiteness.

▶ **Example 5.** The formula $\varphi = \exists_2(1 \in 0)$ ($= \exists_2 X.\ x \in X$ in conventional notation) has two distinct derivatives modulo ACI over the alphabet $\Sigma = bool^1 = [(0), (1)]$ (we abbreviate $\mathsf{fold}\ \delta\ w$ by δ_w).

- $\delta_{[\,]}\ \varphi = \varphi \equiv_{\mathsf{ACI}} \exists_2(1 \in 0 \vee 1 \in 0) = \delta_{[(0)]}\ \varphi \equiv_{\mathsf{ACI}} \delta_{(0)^n}\ \varphi$ for $n > 1$
- $\delta_{[(1)]}\ \varphi = \exists_2(\mathsf{T} \vee \mathsf{F}) \equiv_{\mathsf{ACI}} \exists_2((\mathsf{T} \vee \mathsf{F}) \vee (\mathsf{T} \vee \mathsf{F})) = \delta_{[(1),(0)]}\ \varphi \equiv_{\mathsf{ACI}} \delta_{((0)^n \cdot (1) :: w)}\ \varphi$ for $n : nat,\ w \in \Sigma^*$

5 Accepting Formulas

Defining a function that checks whether a regular expression accepts the empty word is straightforward. A priori, the task seems barely harder for formulas – check whether the empty interpretation $\{\}^n$ (encoded by the empty word) satisfies a formula with n free variables. We start with the following attempt.

$$\begin{aligned} o_< \mathsf{T} &= 1 & o_< (\mathsf{FO}\ x) &= 0 \\ o_< \mathsf{F} &= 0 & o_< (\varphi \vee \psi) &= o_< \varphi \vee o_< \psi \\ o_< (x < y) &= 0 & o_< (\neg\varphi) &= \neg o_< \varphi \\ o_< (x \in X) &= 0 & o_< (\exists_i \varphi) &= o_< \varphi \quad \text{for } i \in \{1, 2\} \end{aligned}$$

As the name of the function indicates this acceptance test will work only for the M2L semantics, justifying M2L's automata theoretic reputation. As we will see the acceptance test for the number theoretic WS1S is by far more involved.

First, we should understand why $o_<$ works well for M2L but fails for WS1S. Therefore, we consider the formula $\varphi = \exists_1 (1 < 0)$ $(= \exists y.\ x < y$ in conventional notation) that behaves differently under the two semantics. Interpreted in WS1S, φ is true: for each natural number x there exists a bigger one. Interpreted in M2L, the fact whether φ is true depends on the interpretation I and the value of x: for example if $\#I = 4$ then the quantifier is allowed to assign to y only values smaller than 4. If additionally $x = 3$, the formula becomes false. When checking acceptance, we only consider interpretations I with $\#I = 0$. Therefore, $o_< \varphi$ rightly returns $\mathsf{0}$ in the M2L setting.

Asserting $o_< (\exists_i \varphi) = o_< \varphi$ effectively corresponds to the restriction of M2L on the quantified values. Fortunately, unrestricted WS1S quantifiers can be related to M2L quantifiers by repeatedly padding the interpretations I with the letter $0_\Sigma = 0^{|I|}$. This means that we can reuse $o_<$ for WS1S if we can get rid of the padding. In the context of automata this step is called futurization and is implemented by traversing the automaton backwards using only 0_Σ-labeled transitions [23]. Here, we work with formulas. Thus traversing means deriving and traversing backwards means deriving from the right. We therefore introduce a second derivative operation $\eth$ (a mirrored δ) that computes the right quotient and is symmetric to the derivative δ. However, the base formulas are not very symmetric themselves, such that the definition of $\eth$ is slightly more complicated than the one for δ. In particular Φ is not closed under $\eth$ – we need to extend it with three further base formulas: $x <_\mathsf{F} y$, $x <_\mathsf{T} y$, and $x \in_\mathsf{T} y$ with the following WS1S semantics (the M2L semantics is the same, but not really relevant).

$$\begin{aligned} I \Vdash (x <_\mathsf{F} y) &= I[x] \neq \{\} \wedge (I[y] = \{\} \vee (I[y] \neq \{\} \wedge \mathsf{Min}\ (I[x]) < \mathsf{Min}\ (I[y]))) \\ I \Vdash (x <_\mathsf{T} y) &= I[y] = \{\} \vee (I[x] \neq \{\} \wedge I[y] \neq \{\} \wedge \mathsf{Min}\ (I[x]) < \mathsf{Min}\ (I[y])) \\ I \Vdash (x \in_\mathsf{T} X) &= I[x] = \{\} \vee (I[x] \neq \{\} \wedge \mathsf{Min}\ (I[x]) \in I[X]) \end{aligned}$$

It is not easy to convey the intuition behind $x <_\mathsf{F} y$, $x <_\mathsf{T} y$, and $x \in_\mathsf{T} X$. In principle, those are random names for formulas denoting "what is left" after deriving from the right. These remainders are closely related to their non-subscripted cousins, behaving differently only if one of the assigned sets is empty. The subscript T indicates the acceptance of the empty model.

Next, we define the right derivative $\eth$. Also the M2L acceptance test $o_<$ must be lifted to the additional formulas. Although we will not use the derivative δ on the new formulas, we want our previously stated theorems still to hold universally. Thus, we extend δ accordingly.

$$
\begin{aligned}
&\eth\ v\ \mathsf{T} &&= \mathsf{T}\\
&\eth\ v\ \mathsf{F} &&= \mathsf{F}\\
&\eth\ v\ (x < y) &&= \begin{cases} x < y & \text{if } \neg v[y] \\ x <_{\mathsf{F}} y & \text{otherwise} \end{cases}\\
&\eth\ v\ (x <_{\mathsf{F}} y) &&= \begin{cases} x <_{\mathsf{T}} y & \text{if } v[x] \wedge \neg v[y] \\ x <_{\mathsf{F}} y & \text{otherwise} \end{cases}\\
&\eth\ v\ (x <_{\mathsf{T}} y) &&= \begin{cases} x <_{\mathsf{T}} y & \text{if } \neg v[y] \\ x <_{\mathsf{F}} y & \text{otherwise} \end{cases}\\
&\eth\ v\ (x \in X) &&= \begin{cases} x \in_{\mathsf{T}} X & \text{if } v[x] \wedge v[X] \\ x \in X & \text{otherwise} \end{cases}\\
&\eth\ v\ (x \in_{\mathsf{T}} X) &&= \begin{cases} x \in X & \text{if } v[x] \wedge \neg v[X] \\ x \in_{\mathsf{T}} X & \text{otherwise} \end{cases}\\
&\eth\ v\ (\mathsf{FO}\ x) &&= \begin{cases} \mathsf{T} & \text{if } v[x] \\ \mathsf{FO}\ x & \text{otherwise} \end{cases}
\end{aligned}
$$

$$
\begin{aligned}
&o_<\ (x <_{\mathsf{F}} y) &&= 0\\
&o_<\ (x <_{\mathsf{T}} y) &&= 1\\
&o_<\ (x \in_{\mathsf{T}} X) &&= 1
\end{aligned}
$$

$$
\delta\ v\ (x <_{\mathsf{F}} y) = \begin{cases} x <_{\mathsf{F}} y & \text{if } \neg v[x] \wedge \neg v[y] \\ \mathsf{T} & \text{if } v[x] \wedge \neg v[y] \\ \mathsf{F} & \text{otherwise} \end{cases}
$$

$$
\delta\ v\ (x <_{\mathsf{T}} y) = \begin{cases} x <_{\mathsf{T}} y & \text{if } \neg v[x] \wedge \neg v[y] \\ \mathsf{T} & \text{if } v[x] \wedge \neg v[y] \\ \mathsf{F} & \text{otherwise} \end{cases}
$$

$$
\delta\ v\ (x \in_{\mathsf{T}} X) = \begin{cases} x \in_{\mathsf{T}} X & \text{if } \neg v[x] \\ \mathsf{T} & \text{if } v[x] \wedge v[X] \\ \mathsf{F} & \text{otherwise} \end{cases}
$$

$$
\begin{aligned}
&\eth\ v\ (\varphi \vee \psi) &&= \eth\ v\ \varphi \vee \eth\ v\ \psi\\
&\eth\ v\ (\neg\varphi) &&= \neg\eth\ v\ \varphi\\
&\eth\ v\ (\exists_i\ \varphi) &&= \exists_i\ (\eth\ (1 :: v)\ \varphi \vee \eth\ (0 :: v)\ \varphi) \quad \text{for } i \in \{1, 2\}
\end{aligned}
$$

A first thing worth noticing is that again the number of derivatives $\eth$ modulo ACI of $\vee$ is finite. The proof is analogous to the proof of Theorem 4.

▶ **Theorem 6.** *A formula φ has finitely many distinct right word derivatives modulo ACI of $\vee$.*

Next, we establish a correspondence between the M2L semantics and right derivatives, similarly to Theorem 3. Therefore, we introduce the operation SNOC, symmetric to CONS, on interpretations.

$$
\begin{aligned}
&\mathsf{SNOC}\ []\ [] = []\\
&\mathsf{SNOC}\ (1 :: v)\ (X :: I) = (\{\#I\} \cup X) :: \mathsf{SNOC}\ v\ I\\
&\mathsf{SNOC}\ (0 :: v)\ (X :: I) = X :: \mathsf{SNOC}\ v\ I
\end{aligned}
$$

Just as for CONS, we require $\#(\mathsf{SNOC}\ v\ I) = \#I + 1$. Another routine induction yields the fundamental property of right derivatives. Note that there is no equivalent theorem for the WS1S semantics.

▶ **Theorem 7.** $I \Vdash_< \eth\ v\ \varphi \Leftrightarrow \mathsf{SNOC}\ v\ I \Vdash_< \varphi$

Finally, we define a function $\mathsf{futurize}$ to repeatedly apply $\eth\ 0_\Sigma$, an operation $\lfloor - \rfloor$ to recursively apply $\mathsf{futurize}$ to all quantifiers in a formula, and the acceptance test o for WS1S formulas. The equations of $\lfloor - \rfloor$ are matched sequentially.

$$
\begin{aligned}
&\mathsf{futurize}\ \varphi =\\
&\quad \text{let } \mathsf{fut}\ \varphi\ X =\\
&\qquad \text{if } \varphi \in X \text{ then } \bigvee X\\
&\qquad \text{else } \mathsf{fut}\ |\eth\ 0_\Sigma\ \varphi|_{\mathsf{ACI}}\ (\{\varphi\} \cup X)\\
&\quad \text{in } \mathsf{fut}\ |\varphi|_{\mathsf{ACI}}\ \{\}
\end{aligned}
\qquad
\begin{aligned}
\lfloor \varphi \vee \psi \rfloor &= \lfloor \varphi \rfloor \vee \lfloor \psi \rfloor\\
\lfloor \neg\varphi \rfloor &= \neg\lfloor \varphi \rfloor\\
\lfloor \exists_i\ \varphi \rfloor &= \mathsf{futurize}\ (\exists_i\ \lfloor \varphi \rfloor)\\
\lfloor \varphi \rfloor &= \varphi\\
o\ \varphi &= o_<\ \lfloor \varphi \rfloor
\end{aligned}
$$

Due to Theorem 6 the function $\mathsf{futurize}$ does always terminate. Moreover, we can prove the expected properties of the acceptance tests.

▶ **Theorem 8.** *Let n be the number of free variables in φ and I an interpretation with $|I| = n$. Then*

1. $o_< \varphi \Leftrightarrow \{\}^n \Vdash_< \varphi$,
2. $I \Vdash_< \mathsf{futurize}\ \varphi \Leftrightarrow \exists k.\ ((\mathsf{SNOC}\ (0^n))^k\ I) \Vdash_< \varphi$,
3. $I \Vdash_< \lfloor \varphi \rfloor \Leftrightarrow I \Vdash \varphi$, *and*
4. $o\ \varphi \Leftrightarrow \{\}^n \Vdash \varphi$,

where $(\mathsf{SNOC}\ (0^n))^k$ *denotes the k-fold repeated application of* $\mathsf{SNOC}\ (0^n)$.

Proof. Prop. 1 follows by routine induction on φ. Prop. 2 follows from the fact that $\mathsf{futurize}\ \varphi$ is the disjunction of all right derivatives of φ by words of the form $(0^n)^k$ for some k and Theorem 7. Prop. 3 is again a routine induction using Prop. 2 in the existential quantifier cases. Finally, for Prop. 4 we calculate: $o\ \varphi \Leftrightarrow o_<\ \lfloor \varphi \rfloor \overset{\text{Prop. 1}}{\Leftrightarrow} \{\}^n \Vdash_< \lfloor \varphi \rfloor \overset{\text{Prop. 3}}{\Leftrightarrow} \{\}^n \Vdash \varphi$. ◀

6 Decision Procedure for WS1S

We are close to being able to instantiate the generic bisimulation computation with our notions to obtain a decision procedure for WS1S. The only missing jigsaw pieces are the treatment of first-order quantification and wellformedness checks on interpretations and formulas that were swept under the carpet so far. Let us make it more precise: in full detail the language of a formula is not just $\mathsf{L}\ \varphi = \{w_I \mid I \Vdash \varphi\}$, but rather has an additional explicit wellformedness condition $\hat{\mathsf{L}}\ \varphi = \{w_I \mid I \vDash \varphi \wedge (\forall x \in \mathsf{FOV}\ \varphi.\ I[x] \neq \{\})\}$ (similarly $\hat{\mathsf{L}}_<\ \varphi = \{w_I \mid I \vDash_< \varphi \wedge (\forall x \in \mathsf{FOV}\ \varphi.\ I[x] \neq \{\})\}$), where $\mathsf{FOV}\ \varphi$ is the set of free first-order variables of φ. Then, the language of the negated formula is

$$\hat{\mathsf{L}}\ (\neg\varphi) = \{w_I \mid \neg(I \vDash \varphi) \wedge (\forall x \in \mathsf{FOV}\ \varphi.\ I[x] \neq \{\})\} \neq \Sigma^* \setminus \hat{\mathsf{L}}\ \varphi.$$

We conclude that complementation on the language level does not correspond to the negation on the formula level. The solution is once again inspired by the treatment of this issue in MONA[3]: certain critical w.r.t. negation wellformedness annotations (critical here means: cannot be checked statically for a formula before starting to derive), called *restrictions*, are syntactically introduced in the formula by the initial transformation ι of bisim. Here, the formula $\mathsf{FO}\ x$ plays another important role, since in our case restrictions merely ensure that only non-empty sets are assigned to first-order variables.

For example, the acceptance test o would cast the formula $\forall_1\ \mathsf{FO}\ 0 = \neg\exists_1\ \neg\mathsf{FO}\ 0$ false without restrictions. However, if we conjoin the additional information that all first-order variables should be non-empty right under their introducing quantifiers and once globally for free first-order variables of a formula, we obtain the desired behavior: $o\ (\neg\exists_1\ \mathsf{FO}\ 0 \wedge \neg\mathsf{FO}\ 0) = 1$. The conjunction of restrictions is implemented by the function $\mathsf{restrict}$ (with equations of the helper function restr matched sequentially and the function FOV denoting the list of free first-order variables of a formula).

$$\begin{aligned}
\mathsf{restr}\ (\varphi \vee \psi) &= \mathsf{restr}\ \varphi \vee \mathsf{restr}\ \psi \\
\mathsf{restr}\ (\neg\varphi) &= \neg\mathsf{restr}\ \varphi \\
\mathsf{restr}\ (\exists_1\ \varphi) &= \exists_1\ (\mathsf{restr}\ \varphi \wedge \mathsf{FO}\ 0) \\
\mathsf{restr}\ (\exists_2\ \varphi) &= \exists_2\ (\mathsf{restr}\ \varphi) \\
\mathsf{restr}\ \varphi &= \varphi \\
\mathsf{restrict}\ \varphi &= \mathsf{fold}\ (\lambda i\ \varphi.\ \varphi \wedge \mathsf{FO}\ i)\ (\mathsf{FOV}\ \varphi)\ (\mathsf{restr}\ \varphi)
\end{aligned}$$

[3] Recent versions of MONA use a more efficient solution than the outlined simple approach [23].

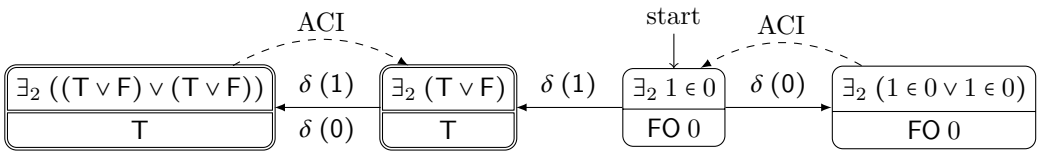


Figure 2 Bisimulation constructed by eqv for $\exists_2 1 \in 0 \equiv \mathsf{FO}\ 0$.

The function restrict translates between L and $\hat{\mathsf{L}}$:

▶ **Theorem 9.** $\mathsf{L}\ (\mathsf{restrict}\ \varphi) = \hat{\mathsf{L}}\ \varphi$ *and* $\mathsf{L}_<\ (\mathsf{restrict}\ \varphi) = \hat{\mathsf{L}}_<\ \varphi$.

Finally, we are ready to instantiate bisim to obtain decision procedures for the M2L and the WS1S semantics. Note that the argument formulas still have to be checked for simple wellformedness properties statically (e.g., by a type system that ensures that no variable is used as both first-order and second-order simultaneously). Also the alphabet Σ must be a list of vectors whose lengths correspond to the number of free variables.

$$\mathsf{eqv}_<\ \Sigma\ \varphi\ \psi = \mathsf{bisim}\ \Sigma\ (\lambda\varphi.\ |\mathsf{restrict}\ \varphi \wedge \mathsf{FO}\ z|_{\mathsf{ACI}})\ (\lambda a\ \varphi.\ |\delta\ a\ \varphi|_{\mathsf{ACI}})\ o_<\ \varphi\ \psi$$
$$\mathsf{eqv}\ \Sigma\ \varphi\ \psi\ \ = \mathsf{bisim}\ \Sigma\ (\lambda\varphi.\ |\mathsf{restrict}\ \varphi|_{\mathsf{ACI}})\ (\lambda a\ \varphi.\ |\delta\ a\ \varphi|_{\mathsf{ACI}})\ o\ \varphi\ \psi$$

The last unexpected thing is the conjunction of a fresh variable z (not occurring in φ and ψ) in the decision procedure for M2L. This is required to circumvent the situation that a M2L formula can only be true in the empty model if it does not quantify over first-order variables and has no free first-order variables. Because of this $\forall_1 \mathsf{FO}\ 0$ would be different from T but equivalent to $\mathsf{FO}\ 42$. Conjoining a fresh first-order variable restriction essentially means, that the first pair of a bisimulation must not be checked for acceptance. This admittedly surprising workaround is required due to the unusual quantification rules of M2L. In WS1S there is nothing special about empty models.

▶ **Theorem 10.** $\mathsf{eqv}_<\ \Sigma\ \varphi\ \psi \Leftrightarrow \hat{\mathsf{L}}_<\ \varphi = \hat{\mathsf{L}}_<\ \psi$ *and* $\mathsf{eqv}\ \Sigma\ \varphi\ \psi \Leftrightarrow \hat{\mathsf{L}}\ \varphi = \hat{\mathsf{L}}\ \psi$

Proof. Using Theorem 2 and discharging its assumptions using the Theorems 3, 4, and 8 together with the characteristic properties of CONS and restrict. ◀

▶ **Example 11.** Figure 2 shows the bisimulation produced by our decision procedure for the equivalent formulas $\varphi = \exists_2 1 \in 0$ and $\psi = \mathsf{FO}\ 0$ over the alphabet $\Sigma = \mathit{bool}^1 = [(1), (0)]$. Graphically, a bisimulation can also be viewed as the product automaton of the automata whose states are derivatives of the formulas with transitions given by δ. Accepting states, i.e., pairs (φ, ψ) for which $o\ \varphi$ and $o\ \psi$ holds, are marked with a double margin. Previously seen ACI equivalent states, detected using the ACI normalization $|-|_{\mathsf{ACI}}$ and marked by a dashed back-edge, are not explored further (and not even checked for being both accepting or both not accepting).

The example suggests that a stronger normalization function than $|-|_{\mathsf{ACI}}$ might be useful. For example, trivial identities involving T or F (e.g., $\mathsf{T} \vee \varphi \equiv \varphi$) should be also simplified. Indeed in our tests we employ a much stronger normalization function, that additionally eliminates quantifiers $\exists_i\ \varphi \equiv \varphi$ provided that 0 is not free in φ. MONA performs a similar optimization, which according to its user manual [24] "can cause tremendous improvements". However, MONA can perform this optimization only on the initially entered formula, while for us it is available for every pair of formulas in the bisimulation because we keep the rich formula structure.

Table 1 Running times (in sec) of regular expression- and formula-based decision procedures.

n	$\psi_n \equiv_{\mathsf{MSO}} \mathsf{T}$ [39]	$\psi_n \equiv_{\mathsf{WS1S}} \mathsf{T}$ [39]	$\mathsf{eqv}_< \psi_n \mathsf{T}$	$\mathsf{eqv}\ \psi_n \mathsf{T}$
0	0	0.4	0	0
1	0	14.4	0	0
2	3.9	–	0	0
10	–	–	0.07	0.07
15	–	–	3.43	3.46

This is only one of a wealth of optimizations performed in MONA. Competing with this state-of-the-art tool is not our main objective yet. Our procedure does outperform MONA on specific examples, e.g., formulas of the form $\varphi \wedge \psi$ where the minimal automata for φ is small, the one for ψ is huge, and for almost all $w \in \Sigma^*$ either $\mathsf{fold}\ \delta\ w\ \varphi = \mathsf{F}$ or $\mathsf{fold}\ \delta\ w\ \psi = \mathsf{F}$. When extending the syntax with formulas of the form $x = n$ for constants n (and defining the left and right derivatives for them), such examples can be immediately constructed: $x = 10 \wedge x = 10000$ takes MONA roughly 40 minutes to disprove, while our procedure requires only a few seconds. MONA computes eagerly both minimal automata for φ and ψ, whereas our procedure is lazier, which pays off here. More generally, we can compete with and outperform by far another existing verified symbolic decision procedure for WS1S, namely the one translating formulas into regular expressions proposed by us earlier [39].

We have evaluated the performance of $\mathsf{eqv}_<$ and eqv for the family of formulas ψ_n from our earlier work [39] using the same hardware and compared it against the best timings obtained for M2L and WS1S in there. The encouraging results are shown in Table **??**. A – means that the computation did not terminate within an hour. MONA solves all those examples instantly. We should note that the optimization $\exists_i\ \varphi \equiv \varphi$ provided that 0 is not free in φ is indeed a tremendous improvement – without its usage our decision procedure scales only slightly better than the regular expression-based one. The possibility to inspect the binder structure after several derivation steps, which is not available when translating to regular expressions, is the main advantage of the proposed decision procedure.

7 Related Work and Discussion

Much of the related work has been discussed earlier. We outline further references which yield some inspiration for generalizations and performance improvement of our algorithm.

For regular expression equivalences different variations of Brzozowski derivatives have been proposed. Most prominently, the partial derivatives have been introduced by Antimirov [1] and generalized by Caron et al. [9] to support complements and intersections. Partial derivatives capture ACI equivalence directly in the data structure of finite sets, eliminating the need for the subsequent ACI normalization after deriving. This idea immediately carries over to formulas and, since decision procedures based on partial derivatives tend to perform better in practice, is worth investigating. Further variations of derivatives exist, however a generalization to complements and intersections seems difficult. In earlier work [30], we provide an overview and a performance comparison.

Beyond regular expressions, there exist generalizations of Brzozowski derivatives for context-free grammars [12, 28] and Kleene algebra with tests (KAT) [26]. In the latter case, derivatives give rise to efficient coalgebraic decision procedures for KAT [31] and its network-targeted specialization NetKAT [15]. Our work can be seen in line with this work, albeit replacing "efficient" with "verified". The ultimate goal is to develop procedures that fulfill both predicates. Below we outline how to possibly improve on efficiency for WS1S.

Our algorithm constructs a bisimulation. It is well known that a bisimulation up to congruence and context is often much smaller. Bonchi and Pous [6] successfully employ this technique in practice, outperforming state-of-the-art solvers for NFA equivalence. Fiedor et al. [14] employ the modern antichain technique for nondeterministic automata in the context of deciding WS1S. Their tool, dWiNA, outperforms MONA for certain classes of formulas (in particular for formulas in prefix normal form). Bonchi and Pous [6] showed that bisimulation up to congruence and context is an even further improvement over antichains for nondeterministic automata. Either of these techniques will improve the performance of our algorithm (extended to work nondeterministically using partial derivatives) at least for some classes of formulas.

MONA benefits a lot from the BDD representation of automata [25]. Pous [31] presents a fast symbolic decision procedure for KAT employing BDDs whose leafs are KAT expressions and lifting derivative operations to those BDDs. We expect this technique to be applicable to WS1S formulas.

An alternative to conjoining formulas $\mathsf{FO}\ 0$ to every first-order quantifier, that turned out to be more efficient in MONA, is to work with a three-valued semantics [23] of acceptance (the third value indicates that no 1 for a first-order variable has been read yet). In our setting, this would require a change of the underlying final coalgebra (essentially working with Moore machines instead of DFAs), and therefore a complication of the syntactic coalgebra.

Ganzow and Kaiser [17, 16] present a very general, model theoretic decision procedure for weak monadic second-order logic on inductive structures that is inspired by Shelah's composition method [33]. Although they tackle the problem from a completely different angle, their computation of the next-state function is also performed symbolically on formulas and therefore has a similar flavor as our left derivatives. However, after obtaining the next-state function in such a way, Ganzow and Kaiser escape the formula world by translating the problem into a finite reachability game. Their algorithm is implemented in the Toss tool. It outperforms MONA for certain formulas, by outsourcing parts of the computation to state-of-the-art SAT-solvers. In contrast, by employing the coalgebraic machinery, we do even more on formulas directly, e.g., by computing the acceptance test via right derivatives. Overall, the work by Ganzow and Kaiser can serve as a starting point on how to generalize our procedure beyond WS1S.

Since S1S, in which second-order variables quantify over potentially infinite sets, is also decidable, the question naturally arises whether our ideas are applicable there as well. Some hope for the affirmative answer can be drawn from the work by Esparza and Kretínský [13], who employ something similar to "derivatives of LTL formulas" (but in conjunction with a non-standard automaton model) for model checking. Furthermore, Ciancia and Venema [10] recast a more standard automaton model on infinite words into the coalgebraic setting.

Methodologically, Isabelle facilitated convenient reasoning about logic and in particular formal languages coalgebraically, not least due to recently introduced codatatype support [5]. The coinductive theory of languages is interesting in itself and formalized separately [37].

8 Conclusion

We have presented a simple coalgebraic algorithm for deciding WS1S. The algorithm operates directly on formulas by deriving them symbolically. The algorithm has been formalized and proved correct in Isabelle/HOL. Using Isabelle's code generator [18], we can extract a fully verified prover for WS1S from the formalization. The formalization, comprising 4000 lines of code, is available online [38]. For readers unfamiliar with Isabelle, a separate manual implementation of the algorithm in Standard ML is also available [38].

In future work, we plan to implement optimizations hinted at in the previous section and to explore derivatives for other weaker and stronger logics such as Presburger Arithmetic, LTL, S1S, WS2S, and S2S following Conway's motto "differentiation is a skill worth acquiring" [11, p. 43].

Acknowledgments. I am grateful to Tobias Nipkow for the continuous encouragement of this work. Damien Pous and several anonymous reviewers helped to improve the paper through a multitude of precious comments. This research is supported by the Deutsche Forschungsgemeinschaft (DFG) program Program and Model Analysis (PUMA, doctorate program 1480).

References

1 Valentin Antimirov. Partial derivatives of regular expressions and finite automaton constructions. *Theor. Comput. Sci.*, 155(2):291–319, March 1996.

2 Abdelwaheb Ayari and David Basin. Bounded model construction for monadic second-order logics. In E. A. Emerson and A. P. Sistla, editors, *CAV 2000*, volume 1855 of *LNCS*, pages 99–112. Springer, 2000.

3 D. Basin and N. Klarlund. Automata based symbolic reasoning in hardware verification. *Formal Methods In System Design*, 13:255–288, 1998. Extended version of: "Hardware verification using monadic second-order logic," *CAV'95*, LNCS 939.

4 Kai Baukus, Saddek Bensalem, Yassine Lakhnech, and Karsten Stahl. Abstracting WS1S systems to verify parameterized networks. In Susanne Graf and Michael I. Schwartzbach, editors, *TACAS 2000*, volume 1785 of *LNCS*, pages 188–203. Springer, 2000.

5 Jasmin Christian Blanchette, Johannes Hölzl, Andreas Lochbihler, Lorenz Panny, Andrei Popescu, and Dmitriy Traytel. Truly modular (co)datatypes for Isabelle/HOL. In Gerwin Klein and Ruben Gamboa, editors, *ITP 2014*, volume 8558 of *LNCS*, pages 93–110. Springer, 2014.

6 Filippo Bonchi and Damien Pous. Checking NFA equivalence with bisimulations up to congruence. In Roberto Giacobazzi and Radhia Cousot, editors, *POPL 2013*, pages 457–468. ACM, 2013.

7 Janusz A. Brzozowski. Derivatives of regular expressions. *J. ACM*, 11(4):481–494, October 1964.

8 J. Richard Büchi. Weak second-order arithmetic and finite automata. *Z. Math. Logik und Grundl. Math.*, 6:66–92, 1960.

9 Pascal Caron, Jean-Marc Champarnaud, and Ludovic Mignot. Partial derivatives of an extended regular expression. In A.-H. Dediu, S. Inenaga, and C. Martín-Vide, editors, *LATA 2011*, volume 6638 of *LNCS*, pages 179–191. Springer, 2011.

10 Vincenzo Ciancia and Yde Venema. Stream automata are coalgebras. In Dirk Pattinson and Lutz Schröder, editors, *CMCS 2012*, volume 7399 of *LNCS*, pages 90–108. Springer, 2012.

11 J.H. Conway. *Regular Algebra and Finite Machines*. Chapman and Hall mathematics series. Dover Publications, Incorporated, 2012.

12 Nils Anders Danielsson. Total parser combinators. In P. Hudak and S. Weirich, editors, *ICFP 2010*, pages 285–296. ACM, 2010.

13 Javier Esparza and Jan Kretínský. From LTL to deterministic automata: A Safraless compositional approach. In Armin Biere and Roderick Bloem, editors, *CAV 2014*, volume 8559 of *LNCS*, pages 192–208. Springer, 2014.

14 Tomáš Fiedor, Lukáš Holík, Ondřej Lengál, and Tomáš Vojnar. Nested antichains for WS1S. In Christel Baier and Cesare Tinelli, editors, *TACAS 2015*, LNCS. Springer, 2015. to appear.

15 Nate Foster, Dexter Kozen, Matthew Milano, Alexandra Silva, and Laure Thompson. A coalgebraic decision procedure for NetKAT. In David Walker, editor, *POPL 2015*, pages 343–355. ACM, 2015.

16 Tobias Ganzow. *Definability and Model Checking: The Role of Orders and Compositionality*. PhD thesis, RWTH Aachen University, 2012.

17 Tobias Ganzow and Lukasz Kaiser. New algorithm for weak monadic second-order logic on inductive structures. In Anuj Dawar and Helmut Veith, editors, *CSL 2010*, volume 6247 of *LNCS*, pages 366–380. Springer, 2010.

18 Florian Haftmann and Tobias Nipkow. Code generation via higher-order rewrite systems. In M. Blume, N. Kobayashi, and G. Vidal, editors, *FLOPS 2010*, volume 6009 of *LNCS*, pages 103–117. Springer, 2010.

19 Jad Hamza, Barbara Jobstmann, and Viktor Kuncak. Synthesis for regular specifications over unbounded domains. In Roderick Bloem and Natasha Sharygina, editors, *FMCAD 2010*, pages 101–109. IEEE, 2010.

20 Jesper G. Henriksen, Jakob L. Jensen, Michael E. Jørgensen, Nils Klarlund, Robert Paige, Theis Rauhe, and Anders Sandholm. MONA: Monadic second-order logic in practice. In E. Brinksma, R. Cleaveland, K. Larsen, T. Margaria, and B. Steffen, editors, *TACAS 1995*, volume 1019 of *LNCS*, pages 89–110. Springer, 1995.

21 Bart Jacobs. A bialgebraic review of deterministic automata, regular expressions and languages. In Kokichi Futatsugi, Jean-Pierre Jouannaud, and José Meseguer, editors, *Algebra, Meaning, and Computation*, volume 4060 of *LNCS*, pages 375–404. Springer, 2006.

22 Jakob L. Jensen, Michael E. Jørgensen, Nils Klarlund, and Michael I. Schwartzbach. Automatic verification of pointer programs using monadic second-order logic. In Marina C. Chen, Ron K. Cytron, and A. Michael Berman, editors, *PLDI 1997*, pages 226–236. ACM, 1997.

23 Nils Klarlund. Relativizations for the logic-automata connection. *Higher-Order and Symbolic Computation*, 18(1-2):79–120, 2005.

24 Nils Klarlund and Anders Møller. *MONA Version 1.4 User Manual*. BRICS, Department of Computer Science, Aarhus University, January 2001. Notes Series NS-01-1. Available from `http://www.brics.dk/mona/`. Revision of BRICS NS-98-3.

25 Nils Klarlund, Anders Møller, and Michael I. Schwartzbach. MONA implementation secrets. *Int. J. Found. Comput. Sci.*, 13(4):571–586, 2002.

26 Dexter Kozen. On the coalgebraic theory of Kleene algebra with tests. Technical Report `http://hdl.handle.net/1813/10173`, Computing and Information Science, Cornell University, March 2008.

27 Albert R. Meyer. Weak monadic second order theory of succesor is not elementary-recursive. In Rohit Parikh, editor, *Logic Colloquium*, volume 453 of *LNM*, pages 132–154. Springer, 1975.

28 Matthew Might, David Darais, and Daniel Spiewak. Parsing with derivatives: A functional pearl. In Manuel M. T. Chakravarty, Zhenjiang Hu, and Olivier Danvy, editors, *ICFP 2011*, pages 189–195. ACM, 2011.

29 Tobias Nipkow, Lawrence Paulson, and Markus Wenzel. *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

30 Tobias Nipkow and Dmitriy Traytel. Unified decision procedures for regular expression equivalence. In G. Klein and R. Gamboa, editors, *ITP 2014*, volume 8558 of *LNCS*, pages 450–466. Springer, 2014.

31 Damien Pous. Symbolic algorithms for language equivalence and Kleene algebra with test. In David Walker, editor, *POPL 2015*, pages 357–368. ACM, 2015.

32 Jan J. M. M. Rutten. Automata and coinduction (an exercise in coalgebra). In D. Sangiorgi and R. de Simone, editors, *CONCUR 1998*, volume 1466 of *LNCS*, pages 194–218. Springer, 1998.

33 Saharon Shelah. The monadic theory of order. *Ann. Math.*, 102(3):379–419, 1975.

34 Wolfgang Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, pages 389–455. Springer, 1997.

35 Dmitriy Traytel. [dWiNA Issue #1] Wrong results for simple formulas, 14 Jan 2015. Archived at `https://github.com/Raph-Stash/dWiNA/issues/1`.

36 Dmitriy Traytel. [Toss-devel] Toss deciding MSO, 14 Jan 2015. Archived at `http://sourceforge.net/p/toss/mailman/message/33232473/`.

37 Dmitriy Traytel. A codatatype of formal languages. In Gerwin Klein, Tobias Nipkow, and Lawrence Paulson, editors, *Archive of Formal Proofs*. `http://afp.sf.net/entries/Coinductive_Languages.shtml`, 2013.

38 Dmitriy Traytel. Supplementary material. `https://github.com/dtraytel/WS1S`, 2015.

39 Dmitriy Traytel and Tobias Nipkow. Verified decision procedures for MSO on words based on derivatives of regular expressions (extended version of [40]). `http://www21.in.tum.de/~traytel/mso.pdf`.

40 Dmitriy Traytel and Tobias Nipkow. Verified decision procedures for MSO on words based on derivatives of regular expressions. In G. Morrisett and T. Uustalu, editors, *ICFP 2013*, pages 3–12. ACM, 2013.

Weak Subgame Perfect Equilibria and their Application to Quantitative Reachability*

Thomas Brihaye[1], Véronique Bruyère[2], Noémie Meunier[†2], and Jean-François Raskin[‡3]

1 **Département de mathématique, Université de Mons (UMONS) Mons, Belgium**
2 **Département d'informatique, Université de Mons (UMONS) Mons, Belgium**
3 **Département d'informatique, Université Libre de Bruxelles (U.L.B.) Brussels, Belgium**

Abstract

We study n-player turn-based games played on a finite directed graph. For each play, the players have to pay a cost that they want to minimize. Instead of the well-known notion of Nash equilibrium (NE), we focus on the notion of subgame perfect equilibrium (SPE), a refinement of NE well-suited in the framework of games played on graphs. We also study natural variants of SPE, named weak (resp. very weak) SPE, where players who deviate cannot use the full class of strategies but only a subclass with a finite number of (resp. a unique) deviation step(s).

Our results are threefold. Firstly, we characterize in the form of a Folk theorem the set of all plays that are the outcome of a weak SPE. Secondly, for the class of quantitative reachability games, we prove the existence of a finite-memory SPE and provide an algorithm for computing it (only existence was known with no information regarding the memory). Moreover, we show that the existence of a constrained SPE, i.e. an SPE such that each player pays a cost less than a given constant, can be decided. The proofs rely on our Folk theorem for weak SPEs (which coincide with SPEs in the case of quantitative reachability games) and on the decidability of MSO logic on infinite words. Finally with similar techniques, we provide a second general class of games for which the existence of a (constrained) weak SPE is decidable.

1998 ACM Subject Classification B.6.3 [Design Aids] Automatic Synthesis, F.1.2 [Modes of Computation] Interactive and Reactive Computation

Keywords and phrases multi-player games on graphs, quantitative objectives, Nash equilibrium, subgame perfect equilibrium, quantitative reachability

Digital Object Identifier 10.4230/LIPIcs.CSL.2015.504

1 Introduction

Two-player zero-sum infinite duration games played on graphs are a mathematical model used to formalize several important problems in computer science. Reactive system synthesis is one such important problem. In this context, see e.g. [14], the vertices and the edges of the graph represent the states and the transitions of the system; one player models the system to synthesize, and the other player models the (uncontrollable) environment of the system.

* This work has been partly supported by European project Cassting (FP7-ICT-601148).
† Author supported by F.R.S.-FNRS fellowship.
‡ Author supported by ERC Starting Grant (279499: inVEST).

24th EACSL Annual Conference on Computer Science Logic (CSL 2015).
Editor: Stephan Kreutzer; pp. 504–518

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In the classical setting, the objectives of the two players are opposite, i.e. the environment is *adversarial*. Modeling the environment as fully adversarial is usually a bold abstraction of reality and there are recent works that consider the more general setting of non zero-sum games which allow to take into account the different objectives of each player. In this latter setting the environment has its own objective which is most often *not* the negation of the objective of the system. The concept of *Nash equilibrium* (NE) [12] is central to the study of non zero-sum games and can be applied to the general setting of n player games. A strategy profile is a NE if no player has an incentive to deviate unilaterally from his strategy, since he cannot strictly improve on the outcome of the strategy profile by changing his strategy only.

However in the context of sequential games (such as games played on graphs), it is well-known that NEs present a serious weakness: a NE allows for *non-credible threats* that rational players should not carry out [16]. Hence, for sequential games, the notion of NE has been strengthened into the notion of *subgame perfect equilibrium* (SPE): a strategy profile is an SPE if it is a NE in all the subgames of the original game. While the notion of SPE is rather well understood for finite state game graphs with ω-regular objectives or for games in finite extensive form (finite game trees), less is known for game graphs with *quantitative objectives* in which players encounter costs that they want to minimize, like in classical quantitative objectives such as mean-payoff, discounted sum, or quantitative reachability.

Several natural and important questions arise for such games: Can we decide the existence of an SPE, and more generally the *constrained* existence of an SPE (i.e. an SPE in which each player encounters a cost less than some fixed value)? Can we compute such SPEs that use finite-memory strategies only? Whereas several authors have studied what the hypotheses are to impose on games in a way to guarantee the existence of an SPE, the previous algorithmic questions are still wide open. In this article, we provide progress in the understanding of the notion of SPE. We study some variants of SPEs and establish a theorem that characterizes their possible outcomes in quantitative games. We derive from this characterization interesting algorithms and information on the strategies for two important classes of quantitative games. Our contributions are detailed in the next paragraph.

Contributions. First, we formalize a notion of *deviation step* from a strategy profile that allows us to define two natural variants of NEs. While a NE must be resistant to the unilateral deviation of one player for any number of deviation steps, a *weak* (resp. *very weak*) NE must be resistant to the unilateral deviation of one player for any *finite* number of (resp. a unique) deviation step(s). Then we use those variants to define the corresponding notions of *weak* and *very weak* SPE. The latter notion is very close to the one-step deviation property [13]. Any very weak SPE is also a weak SPE, and there are games for which there exists a weak SPE but no SPE. Also, for games with upper-semicontinuous cost functions and for games played on finite game trees, the three notions are equivalent.

Second, we characterize in the form of a Folk theorem[1] all the possible outcomes of weak SPEs. The characterization is obtained starting from all possible plays of the game and the application of a nonincreasing operator that removes plays that cannot be outcome of a weak SPE. We show that the limit of the nonincreasing chain of sets always exists and contains exactly all the possible outcomes of weak SPEs. Furthermore, we show how for each such outcome, we can associate a strategy profile that generates it and which is a weak SPE.

[1] We do not consider our result as folklore, but we use this terminology, as also done in [6], in reference to the "classical folk theorems" for repeated games which characterize the payoff profiles of NEs and SPEs in repeated games (see for instance Chapter 8 in [13]).

Additionally, to illustrate the potential of our Folk theorem, we show how it can be refined and used to answer open questions about two classes of quantitative games. The first class of games that we consider are *quantitative reachability games*, such that each player aims at reaching his own set of target states as soon as possible. As the cost functions in those games are continuous, our Folk theorem characterizes precisely the outcomes of SPEs and not only weak SPEs. In [1, 7], it has been shown that quantitative reachability games always have SPEs. The proof provided for this theorem is non constructive since it relies on topological arguments. Here, we strengthen this existential result by proving that there always exists, not only an SPE but, a *finite-memory* SPE. Furthermore, we provide an algorithm to construct such a finite memory SPE. This algorithm is based on a constructive version of our Folk Theorem for the class of quantitative reachability games: we show that the nonincreasing chain of sets of potential outcomes stabilizes after a finite number of steps and that each intermediate set is an ω-regular set that can be effectively described using MSO sentences. The second class of games that we consider is the class of games with cost functions that are *prefix-independent*, whose range of values is *finite*, and for which each value has an *ω-regular* pre-image. For this general class of games, with similar techniques as for quantitative reachability games, we show how to construct an effective representation of all possible outcomes compatible with a weak SPE, and consequently that the existence of a weak SPE is decidable. In those two applications, we show that our construction also allow us to answer the question of existence of a constrained (weak) SPE, i.e. a (weak) SPE in which players pays a cost which is bounded by a given value.

Related work. The concept of SPE has been first introduced and studied by the game theory community. The notion of SPE has been first introduced by Kuhn in finite extensive form games [10]. For such games, backward induction can be used to prove that there always exists an SPE. By inspecting the backward induction proof, it is not difficult to realize that the notion of very weak SPE and SPE are equivalent in this context.

SPEs for infinite trees defined as the unfolding of finite graphs with *qualitative*, i.e. win-lose, ω-regular objectives, have been studied by Ummels in [19]: it is proved that such games always have an SPE, and that the existence of a constrained SPE is decidable.

In [9], the authors provide an effective representation of the outcomes of NEs in concurrent priced games by constructing a Büchi automaton accepting the language of outcomes of all NEs satisfying a bound vector. The existence of NEs in quantitative games played on graphs is studied in [3]; it is shown that for a large class of games, there always exists a finite-memory NE. This result is extended in [4] for two-player games and secure equilibria (a refinement of NEs); additionally the constrained existence problem for secure equilibria is also shown decidable for a large range of cost functions. None of these articles consider SPEs.

In [6], the authors prove that for quantitative games with cost functions that are upper-semicontinous and with finite range, there always exists an SPE. This result also relies on a nonincreasing chain of sets of possible outcomes of SPEs. The main differences with our work is that we obtain a Folk theorem that *characterizes* all possible outcomes of weak SPEs with no restriction on the cost functions. Moreover we have shown that our Folk theorem can be made effective for two classes of quantitative games of interest. Effectiveness issues are not considered in [6]. Prior to this work, Mertens shows in [11] that if the cost functions are bounded and Borel measurable then there always exists an ϵ-NE. In [7], Fudenberg et al. show that if the cost functions are all continuous, then there always exists an SPE. Those results were recently extended in [15] by Le Roux and Pauly.

Organization of the article. In Section 2, we present the notions of quantitative game, classical NE and SPE, and their variants. In Section 3, we propose our Folk Theorem for weak SPEs. In Section 4, we provide an algorithm for computing a finite-memory SPE for quantitative reachability games, and a second algorithm to decide the constrained existence of an SPE for this class of games. We also show that the existence of a (constrained) weak SPE is decidable for another class of games. A conclusion and future work are given in the last section.

2 Preliminaries and Variants of Equilibria

In this section, we recall the notions of quantitative game, Nash equilibrium, and subgame perfect equilibrium. We also introduce variants of Nash and subgame perfect equilibria, and compare them with the classical notions.

2.1 Quantitative Games

We consider multi-player turn-based non zero-sum quantitative games in which, for each infinite play, players pay a cost that they want to minimize.[2]

▶ **Definition 1.** A *quantitative game* is a tuple $\mathcal{G} = (\Pi, V, (V_i)_{i\in\Pi}, E, \bar{\lambda})$ where:
- Π is a finite set of players,
- V is a finite set of vertices,
- $(V_i)_{i\in\Pi}$ is a partition of V such that V_i is the set of vertices controlled by player $i \in \Pi$,
- $E \subseteq V \times V$ is a set of edges, such that[3] for all $v \in V$, there exists $v' \in V$ with $(v, v') \in E$,
- $\bar{\lambda} = (\lambda_i)_{i\in\Pi}$ is a cost function such that $\lambda_i : V^\omega \to \mathbb{R} \cup \{+\infty\}$ is player i cost function.

A *play* of $\mathcal{G}$ is an infinite sequence $\rho = \rho_0\rho_1 \ldots \in V^\omega$ such that $(\rho_i, \rho_{i+1}) \in E$ for all $i \in \mathbb{N}$. *Histories* of $\mathcal{G}$ are finite sequences $h = h_0 \ldots h_n \in V^+$ defined in the same way. The *length* $|h|$ of h is the number n of its edges. We denote by $\mathrm{First}(h)$ (resp. $\mathrm{Last}(h)$) the first vertex h_0 (resp. last vertex h_n) of h. Usually histories are non-empty, but in specific situations it will be useful to consider the empty history ϵ. The set of all histories (ended by a vertex in V_i) is denoted by Hist (by Hist_i). A *prefix* (resp. *suffix*) of a play ρ is a finite sequence $\rho_0 \ldots \rho_n$ (resp. infinite sequence $\rho_n\rho_{n+1} \ldots$) denoted by $\rho_{\leq n}$ or $\rho_{<n+1}$ (resp. $\rho_{\geq n}$). We use notation $h < \rho$ when a history h is prefix of a play ρ. Given two distinct plays ρ and ρ', their longest common prefix is denoted by $\rho \wedge \rho'$.

When an initial vertex $v_0 \in V$ is fixed, we call $(\mathcal{G}, v_0)$ an *initialized* quantitative game. A play (resp. a history) of $(\mathcal{G}, v_0)$ is a play (resp. history) of $\mathcal{G}$ starting in v_0. The set of histories $h \in \mathrm{Hist}$ (resp. $h \in \mathrm{Hist}_i$) with $\mathrm{First}(h) = v_0$ is denoted by $\mathrm{Hist}(v_0)$ (resp. $\mathrm{Hist}_i(v_0)$). In the figures of this article, we will often *unravel* the graph of the game $(\mathcal{G}, v_0)$ from the initial vertex v_0, which ends up in an infinite tree.

Given a play $\rho \in V^\omega$, its *cost* is given by $\bar{\lambda}(\rho) = (\lambda_i(\rho))_{i\in\Pi}$. In this article, we are particularly interested in quantitative reachability games in which $\lambda_i(\rho)$ is equal to the number of edges to reach a given set of vertices.

▶ **Definition 2.** A *quantitative reachability game* is a quantitative game $\mathcal{G}$ such that the cost function $\bar{\lambda} : V^\omega \to (\mathbb{N} \cup \{+\infty\})^\Pi$ is defined as follows. Each player i has a *target set* $T_i \subseteq V$, and for each play $\rho = \rho_0\rho_1 \ldots$ of $\mathcal{G}$, the cost $\lambda_i(\rho)$ is the least index n such that $\rho_n \in T_i$ if it exists, and $+\infty$ otherwise.

[2] Alternatively, players could receive a payoff that they want to maximize.
[3] Each vertex has at least one outgoing edge.

Notice that the cost function $\bar{\lambda}$ of a quantitative game is often defined from $|\Pi|$-tuples of weights labeling the edges of the game. For instance, in inf games, $\lambda_i(\rho)$ is equal to the infimum of player i weights seen along ρ. Some other classical examples are liminf, limsup, mean-payoff, and discounted sum games [5]. In case of quantitative reachability on graphs with weighted edges, the cost $\lambda_i(\rho)$ for player i is replaced by the sum of the weights seen along ρ until his target set is reached. We do not consider this extension here. Notice that when weights are positive integers, replacing each edge with cost c by a path of length c composed of c new edges allows to recover Definition 2.

Let us recall the notions of prefix-independent, continuous, and lower- (resp. upper-) semicontinuous cost functions. Since V is endowed with the discrete topology, and thus V^ω with the product topology, a sequence of plays $(\rho_n)_{n\in\mathbb{N}}$ converges to a play $\rho = \lim_{n\to\infty} \rho_n$ if every prefix of ρ is prefix of all ρ_n except, possibly, of finitely many of them.

▶ **Definition 3.** Let λ_i be a player i cost function. Then

- λ_i is *prefix-independent* if $\lambda_i(h\rho) = \lambda_i(\rho)$ for any history h and play ρ.
- λ_i is *continuous* if whenever $\lim_{n\to\infty} \rho_n = \rho$, then $\lim_{n\to\infty} \lambda_i(\rho_n) = \lambda_i(\rho)$.
- λ_i *upper-semicontinuous* (resp. *lower-semicontinuous*) if whenever $\lim_{n\to\infty} \rho_n = \rho$, then $\limsup_{n\to\infty} \lambda_i(\rho_n) \leq \lambda_i(\rho)$ (resp. $\liminf_{n\to\infty} \lambda_i(\rho_n) \geq \lambda_i(\rho)$).

For instance, the cost functions used in liminf and mean-payoff games are prefix-independent, contrarily to the case of inf games. Clearly, if λ_i is continuous, then it is upper- and lower-semicontinuous. The cost functions of liminf and mean-payoff games are neither upper-semicontinuous nor lower-semicontinuous, whereas they are continuous in discounted sum games. The cost functions λ_i used in quantitative reachability games can be transformed into continuous ones as follows [1]: $\lambda'_i(\rho) = 1 - \frac{1}{\lambda_i(\rho)+1}$ if $\lambda_i(\rho) < +\infty$, and $\lambda'_i(\rho) = 1$ otherwise.

2.2 Strategies and Deviations

A *strategy* σ for player $i \in \Pi$ is a function $\sigma : \mathrm{Hist}_i \to V$ assigning to each history[4] $hv \in \mathrm{Hist}_i$ a vertex $v' = \sigma(hv)$ such that $(v, v') \in E$. In an initialized game $(\mathcal{G}, v_0)$, σ is restricted to histories starting with v_0. A player i strategy σ is *positional* if it only depends on the last vertex of the history, i.e. $\sigma(hv) = \sigma(v)$ for all $hv \in \mathrm{Hist}_i$. It is a *finite-memory* strategy if it needs only finite memory of the history (recorded by a finite strategy automaton, also called a Moore machine). A play ρ is *consistent* with a player i strategy σ if $\rho_{k+1} = \sigma(\rho_{\leq k})$ for all k such that $\rho_k \in V_i$. A *strategy profile* of $\mathcal{G}$ is a tuple $\bar{\sigma} = (\sigma_i)_{i\in\Pi}$ of strategies, where each σ_i is a player i strategy. It is called *positional* (resp. *finite-memory*) if all σ_i, $i \in \Pi$, are positional (resp. finite-memory). Given an initial vertex v_0, such a strategy profile determines a unique play of $(\mathcal{G}, v_0)$ that is consistent with all the strategies. This play is called the *outcome* of $\bar{\sigma}$ and is denoted by $\langle\bar{\sigma}\rangle_{v_0}$.

Given σ_i a player i strategy, we say that player i *deviates* from σ_i if he does not stick to σ_i and prefers to use another strategy σ'_i. Let $\bar{\sigma}$ be a strategy profile. When all players stick to their strategy σ_i except player i that shifts to σ'_i, we denote by (σ'_i, σ_{-i}) the derived strategy profile, and by $\langle\sigma'_i, \sigma_{-i}\rangle_{v_0}$ its outcome in $(\mathcal{G}, v_0)$. In the next definition, we introduce the notion of deviation step of a strategy σ'_i from a given strategy profile $\bar{\sigma}$.

[4] In this article we often write a history in the form hv with $v \in V$ to emphasize that v is the last vertex of this history.

▶ **Definition 4.** Let $(\mathcal{G}, v_0)$ be an initialized game, $\bar{\sigma}$ be a strategy profile, and σ'_i be a player i strategy. We say that σ'_i has a *hv-deviation step* from $\bar{\sigma}$ for some history $hv \in \mathrm{Hist}_i(v_0)$ with $v \in V_i$, if $hv < \langle \sigma'_i, \sigma_{-i} \rangle_{v_0}$ and $\sigma_i(hv) \neq \sigma'_i(hv)$.

Notice that the previous definition requires that hv is a prefix of the outcome $\langle \sigma'_i, \sigma_{-i} \rangle_{v_0}$; it says nothing about σ'_i outside of this outcome. A strategy σ'_i can have a finite or an infinite number of deviation steps in the sense of Definition 4. A strategy with three deviation steps is depicted in Figure 1 such that each $h_k v_k$-deviation step from $\bar{\sigma}$, $1 \leq k \leq 3$, is highlighted with a dashed edge.

In light of Definition 4, we introduce the following classes of strategies.

▶ **Definition 5.** Let $(\mathcal{G}, v_0)$ be an initialized game, and $\bar{\sigma}$ be a strategy profile.

- A strategy σ'_i is *finitely deviating* from $\bar{\sigma}$ if it has a finite number of deviation steps from $\bar{\sigma}$.
- It is *one-shot deviating* from $\bar{\sigma}$ if it has a v_0-deviation step from $\bar{\sigma}$, and no other deviation step.

In other words, a strategy σ'_i is finitely deviating from $\bar{\sigma}$ if there exists a history $hv < \langle \sigma'_i, \sigma_{-i} \rangle_{v_0}$ such that for all $h'v'$, $hv \leq h'v' < \langle \sigma'_i, \sigma_{-i} \rangle_{v_0}$, we have $\sigma'_i(h'v') = \sigma_i(h'v')$ (σ'_i acts as σ_i from hv along $\langle \sigma'_i, \sigma_{-i} \rangle_{v_0}$). The strategy σ'_i is one-shot deviating from $\bar{\sigma}$ if it differs from σ_i at the initial vertex v_0, and after v_0 acts as σ_i along $\langle \sigma'_i, \sigma_{-i} \rangle_{v_0}$. As for Definition 4, the previous definition says nothing about σ'_i outside of $\langle \sigma'_i, \sigma_{-i} \rangle_{v_0}$. Clearly any one-shot deviating strategy is finitely deviating. The strategy of Figure 1 is finitely deviating but not one-shot deviating.

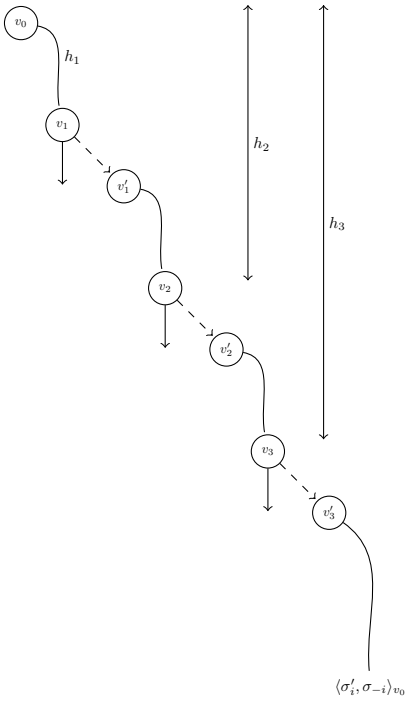


Figure 1 A strategy σ'_i with a finite number of deviation steps.

2.3 Nash and Subgame Perfect Equilibria, and Variants

In this paper, we focus on subgame perfect equilibria and their variants. Let us first recall the classical notion of Nash equilibrium. A strategy profile $\bar{\sigma}$ in an initialized game is a Nash equilibrium if no player has an incentive to deviate unilaterally from his strategy, since he cannot strictly decrease his cost when using any other strategy.

▶ **Definition 6.** Given an initialized game $(\mathcal{G}, v_0)$, a strategy profile $\bar{\sigma} = (\sigma_i)_{i \in \Pi}$ of $(\mathcal{G}, v_0)$ is a *Nash equilibrium (NE)* if for all players $i \in \Pi$, for all player i strategies σ'_i, we have $\lambda_i(\langle \sigma'_i, \sigma_{-i} \rangle_{v_0}) \geq \lambda_i(\langle \bar{\sigma} \rangle_{v_0})$.

We say that a player i strategy σ'_i is a *profitable deviation* for i w.r.t. $\bar{\sigma}$ if $\lambda_i(\langle \sigma'_i, \sigma_{-i} \rangle_{v_0}) < \lambda_i(\langle \bar{\sigma} \rangle_{v_0})$. Therefore $\bar{\sigma}$ is a NE if no player has a profitable deviation w.r.t. $\bar{\sigma}$.

Let us propose the next variants of NE.

▶ **Definition 7.** Let $(\mathcal{G}, v_0)$ be an initialized game. A strategy profile $\bar{\sigma}$ is a *weak NE* (resp. *very weak NE*) in $(\mathcal{G}, v_0)$ if, for each player $i \in \Pi$, for each finitely deviating (resp. one-shot deviating) strategy σ'_i of player i, we have $\lambda_i(\langle \sigma'_i, \sigma_{-i} \rangle_{v_0}) \geq \lambda_i(\langle \bar{\sigma} \rangle_{v_0})$.

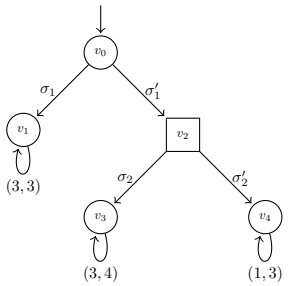


Figure 2 A simple two-player quantitative game.

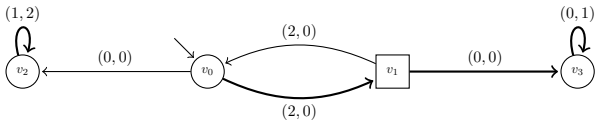


Figure 3 A two-player game with a (very) weak SPE and no SPE. For each player, the cost of a play is his unique weight seen in the ending cycle.

▶ **Example 8.** Consider the two-player quantitative game depicted in Figure 2. Circle (resp. square) vertices are player 1 (resp. player 2) vertices. The edges are labeled by couples of weights such that weights $(0,0)$ are not specified. For each player i, the cost $\lambda_i(\rho)$ of a play ρ is the weight of its ending loop. In this simple game, each player i have two positional strategies that are respectively denoted by σ_i and σ'_i (see Figure 2).

The strategy profile (σ_1, σ'_2) is not a NE since σ'_1 is a profitable deviation for player 1 w.r.t. (σ_1, σ'_2) (player 1 pays cost 1 instead of cost 3). This strategy profile is neither a weak NE nor a very weak NE because in this simple game, player 1 can only deviate from σ_1 by using the one-shot deviating strategy σ'_1. On the contrary, the strategy profile (σ_1, σ_2) is a NE with outcome $v_0 v_1^\omega$ of cost $(3,3)$. It is also a weak NE and a very weak NE.

By definition, any NE is a weak NE, and any weak NE is a very weak NE. The contrary is false: in the previous example, (σ'_1, σ_2) is a very weak NE, but not a weak NE. We will see later an example of game with a weak NE that is not an NE (see Example 12).

The notion of subgame perfect equilibrium is a refinement of NE. In order to define it, we need to introduce the following notions. Given a quantitative game $\mathcal{G} = (\Pi, V, (V_i)_{i\in\Pi}, E, \bar{\lambda})$ and a history h of $\mathcal{G}$, we denote by $\mathcal{G}_{\restriction h}$ the game $\mathcal{G}_{\restriction_h} = (\Pi, V, (V_i)_{i\in\Pi}, E, \bar{\lambda}_{\restriction_h})$ where $\bar{\lambda}_{\restriction h}(\rho) = \bar{\lambda}(h\rho)$ for any play of $\mathcal{G}_{\restriction h}$[5], and we say that $\mathcal{G}_{\restriction h}$ is a *subgame* of $\mathcal{G}$. Given an initialized game $(\mathcal{G}, v_0)$, and a history $hv \in \mathrm{Hist}(v_0)$, the initialized game $(\mathcal{G}_{\restriction h}, v)$ is called the subgame of $(\mathcal{G}, v_0)$ with history hv. Notice that $(\mathcal{G}, v_0)$ can be seen as a subgame of itself with history hv_0 such that $h = \epsilon$. Given a player i strategy σ in $(\mathcal{G}, v_0)$, we define the strategy $\sigma_{\restriction h}$ in $(\mathcal{G}_{\restriction h}, v)$ as $\sigma_{\restriction h}(h') = \sigma(hh')$ for all histories $h' \in \mathrm{Hist}_i(v)$. Given a strategy profile $\bar{\sigma} = (\sigma_i)_{i\in\Pi}$, we use notation $\bar{\sigma}_{\restriction h}$ for $(\sigma_{i\restriction h})_{i\in\Pi}$, and $\langle \bar{\sigma}_{\restriction h} \rangle_v$ is its outcome in the subgame $(\mathcal{G}_{\restriction h}, v)$.

We can now recall the classical notion of subgame perfect equilibrium: it is a strategy profile in an initialized game that induces a NE in each of its subgames. In particular, a subgame perfect equilibrium is a NE.

▶ **Definition 9.** Given an initialized game $(\mathcal{G}, v_0)$, a strategy profile $\bar{\sigma}$ of $(\mathcal{G}, v_0)$ is a *subgame perfect equilibrium (SPE)* if $\bar{\sigma}_{\restriction h}$ is a NE in $(\mathcal{G}_{\restriction h}, v)$, for every history $hv \in \mathrm{Hist}(v_0)$.

As for NE, we propose the next variants of SPE.

▶ **Definition 10.** Let $(\mathcal{G}, v_0)$ be an initialized game. A strategy profile $\bar{\sigma}$ is a *weak SPE* (resp. *very weak SPE*) if $\bar{\sigma}_{\restriction h}$ is a weak NE (resp. very weak NE) in $(\mathcal{G}_{\restriction h}, v)$, for all histories $hv \in \mathrm{Hist}(v_0)$.

[5] In this article, we will always use notation $\bar{\lambda}(h\rho)$ instead of $\bar{\lambda}_{\restriction h}(\rho)$.

▶ **Example 11.** We come back to the game depicted in Figure 2. We have seen before that the strategy profile (σ_1, σ_2) is a NE. Notice that this NE uses a non-credible threat of player 2 that prefers to pay a cost of 4 instead of 3 (by using σ_2'). Such a threat is not allowed for SPEs. Indeed consider the subgame $(\mathcal{G}_{\restriction v_0}, v_2)$ of $(\mathcal{G}, v_0)$ with history v_0v_2. In this subgame, σ_2' is a profitable deviation for player 2, showing that (σ_1, σ_2) is not an SPE. One can easily verify that the strategy profile (σ_1', σ_2') is an SPE, as well as a weak SPE and a very weak SPE, due to the simple form of the game.

The previous example is too simple to show the differences between classical SPEs and their variants. The next example presents a game with a (very) weak SPE but no SPE.

▶ **Example 12.** Consider the two-player game $(\mathcal{G}, v_0)$ in Figure 3. The edges are labeled by couples of weights, and for each player i the cost $\lambda_i(\rho)$ of a play ρ is the unique weight seen in its ending cycle. With this definition, $\lambda_i(\rho)$ can also be seen as either the mean-payoff, or the liminf, or the limsup, of the weights of ρ. It is known that this game has no SPE [17].

Let us show that the strategy profile $\bar{\sigma}$ depicted with thick edges is a very weak SPE. Due to the simple form of the game, only two cases are to be treated. Consider the subgame $(\mathcal{G}_{\restriction h}, v_0)$ with $h \in (v_0v_1)^*$, and the one-shot deviating strategy σ_1' of player 1 such that $\sigma_1'(v_0) = v_2$. Then $\langle \bar{\sigma}_{\restriction h} \rangle_{v_0} = v_0v_1v_3^\omega$ and $\langle \sigma_1', \sigma_{2\restriction h} \rangle_{v_0} = v_0v_2^\omega$, showing that σ_1' is not a profitable deviation for player 1. One also checks that in the subgame $(\mathcal{G}_{\restriction h}, v_1)$ with $h \in (v_0v_1)^*v_0$, the one-shot deviating strategy σ_2' of player 2 such that $\sigma_2'(v_1) = v_0$ is not profitable for him.

Similarly, one can prove that $\bar{\sigma}$ is a weak SPE (see also Proposition 13 hereafter). Notice that $\bar{\sigma}$ is not an SPE. Indeed the strategy σ_2' such that $\sigma_2'(hv_1) = v_0$ for all h, is a profitable deviation for player 2 in $(\mathcal{G}, v_0)$. This strategy is (of course) not finitely deviating. Finally notice that $\bar{\sigma}$ is a weak NE that is not an NE.

From Definition 10, any SPE is a weak SPE, and any weak SPE is a very weak SPE. The next proposition states that weak SPE and very weak SPE are equivalent notions, but this is no longer true for SPE and weak SPE as shown previously by Example 12. The first part of the proof is based on arguments from the one-step deviation property used to prove Kuhn's theorem [10]. The second part follows from Example 12 [17].

▶ **Proposition 13.**

- *Let $(\mathcal{G}, v_0)$ be an initialized game, and $\bar{\sigma}$ be a strategy profile. Then $\bar{\sigma}$ is a weak SPE iff $\bar{\sigma}$ is a very weak SPE.*
- *There exists an initialized game $(\mathcal{G}, v_0)$ with a weak SPE but no SPE.*

Under the next hypotheses on the game or the costs, the equivalence between SPE, weak SPE, and very weak SPE holds. The first case, when the cost functions are continuous, is a classical result in game theory, see for instance [8]; the second case appears as a part of the proof of Kuhn's theorem [10].

▶ **Proposition 14.** *Let $(\mathcal{G}, v_0)$ be an initialized game, and $\bar{\sigma}$ be a strategy profile.*

- *If all cost functions λ_i are continuous, or even upper-semicontinuous[6], then $\bar{\sigma}$ is an SPE iff $\bar{\sigma}$ is a weak SPE iff $\bar{\sigma}$ is a very weak SPE.*
- *If $\mathcal{G}$ is a finite tree[7], then $\bar{\sigma}$ is an SPE iff $\bar{\sigma}$ is a weak SPE iff $\bar{\sigma}$ is a very weak SPE.*

[6] In games where the players receive a payoff that they want to maximize, the hypothesis of upper-semicontinuity has to be replaced by lower-semicontinuity.

[7] In a finite tree game, the plays are finite sequences of vertices ending in a leaf and their cost is associated with the ending leaf. An example of such a game is depicted in Figure 2.

Recall that discounted sum games and quantitative reachability games are continuous. Thus for these games, the three notions of SPE, weak SPE and very weak SPE, are equivalent.

▶ **Corollary 15.** *Let $(\mathcal{G}, v_0)$ be an initialized quantitative reachability game, and $\bar{\sigma}$ be a strategy profile. Then $\bar{\sigma}$ is an SPE iff $\bar{\sigma}$ is a weak SPE iff $\bar{\sigma}$ is a very weak SPE.*

On the opposite, the initialized game of Figure 3 has a weak SPE but no SPE. Its cost function λ_2 is not upper-semicontinuous. Indeed, we have that $\lim_{n\to\infty}(v_0v_1)^n v_3^\omega = (v_0v_1)^\omega$ and $\lim_{n\to\infty} \lambda_2((v_0v_1)^n v_3^\omega) = 1 > 0 = \lambda_2((v_0v_1)^\omega)$.

3 Folk Theorem for Weak SPEs

In this section, we characterize in the form of a Folk Theorem the set of all outcomes of weak SPEs. To this end we define a nonincreasing sequence of sets of plays that initially contain all the plays, and then lose, step by step, some plays that for sure are not outcomes of a weak SPE, until finally reaching a fixpoint.

Let $(\mathcal{G}, v_0)$ be a game. For an ordinal α and a history $hv \in \mathrm{Hist}(v_0)$, let us consider the set $\mathbf{P}_\alpha(hv) = \{\rho \mid \rho \text{ is a } \textit{potential} \text{ outcome of a weak NE in } (\mathcal{G}_{\restriction h}, v) \text{ at step } \alpha\}$. This set is defined by induction on α as follows:

▶ **Definition 16.** Let $(\mathcal{G}, v_0)$ be a quantitative game. The set $\mathbf{P}_\alpha(hv)$ is defined as follows for each ordinal α and history $hv \in \mathrm{Hist}(v_0)$:

- For $\alpha = 0$,
$$\mathbf{P}_\alpha(hv) = \{\rho \mid \rho \text{ is a play in } (\mathcal{G}_{\restriction h}, v)\}. \tag{1}$$
- For a successor ordinal $\alpha + 1$,
$$\mathbf{P}_{\alpha+1}(hv) = \mathbf{P}_\alpha(hv) \setminus \mathbf{E}_\alpha(hv) \tag{2}$$
such that $\rho \in \mathbf{E}_\alpha(hv)$ (see Figure 4) iff
  - there exists a history h', $hv \leq h' < h\rho$, and $\mathrm{Last}(h') \in V_i$ for some i,
  - there exists a vertex v', $h'v' \not< h\rho$,
  - such that $\forall \rho' \in \mathbf{P}_\alpha(h'v')$: $\lambda_i(h\rho) > \lambda_i(h'\rho')$.
- For a limit ordinal α:
$$\mathbf{P}_\alpha(hv) = \bigcap_{\beta<\alpha} \mathbf{P}_\beta(hv). \tag{3}$$

Notice that an element ρ of $\mathbf{P}_\alpha(hv)$ is a play in $(\mathcal{G}_{\restriction h}, v)$ (and not in $(\mathcal{G}, v_0)$). Therefore it starts with vertex v, and $h\rho$ is a play in $(\mathcal{G}, v_0)$. For $\alpha + 1$ being a successor ordinal, play $\rho \in \mathbf{E}_\alpha(hv)$ is erased from $\mathbf{P}_\alpha(hv)$ when for all $\rho' \in \mathbf{P}_\alpha(h'v')$, player i pays a lower cost $\lambda_i(h'\rho') < \lambda_i(h\rho)$, meaning that ρ is no longer a potentiel outcome of a weak NE in $(\mathcal{G}_{\restriction h}, v)$.

The sequence $(\mathbf{P}_\alpha(hv))_\alpha$ is nonincreasing by definition, and reaches a fixpoint:

▶ **Proposition 17.** *There exists an ordinal α_* such that $\mathbf{P}_{\alpha_*}(hv) = \mathbf{P}_{\alpha_*+1}(hv)$ for all histories $hv \in \mathrm{Hist}(v_0)$.*

In the sequel, α_* always refers to the ordinal mentioned in Proposition 17.

Our Folk Theorem for weak SPEs is the next one.

▶ **Theorem 18.** *Let $(\mathcal{G}, v_0)$ be a quantitative game. There exists a weak SPE in $(\mathcal{G}, v_0)$ with outcome ρ iff $\mathbf{P}_{\alpha_*}(hv) \neq \emptyset$ for all $hv \in \mathrm{Hist}(v_0)$, and $\rho \in \mathbf{P}_{\alpha_*}(v_0)$.*

The proof of Theorem 18 follows from Lemmas 19 and 20.

▶ **Lemma 19.** *If $(\mathcal{G}, v_0)$ has a weak SPE $\bar{\sigma}$, then $\mathbf{P}_{\alpha_*}(hv) \neq \emptyset$ for all $hv \in \mathrm{Hist}(v_0)$, and $\langle\bar{\sigma}\rangle_{v_0} \in \mathbf{P}_{\alpha_*}(v_0)$.*

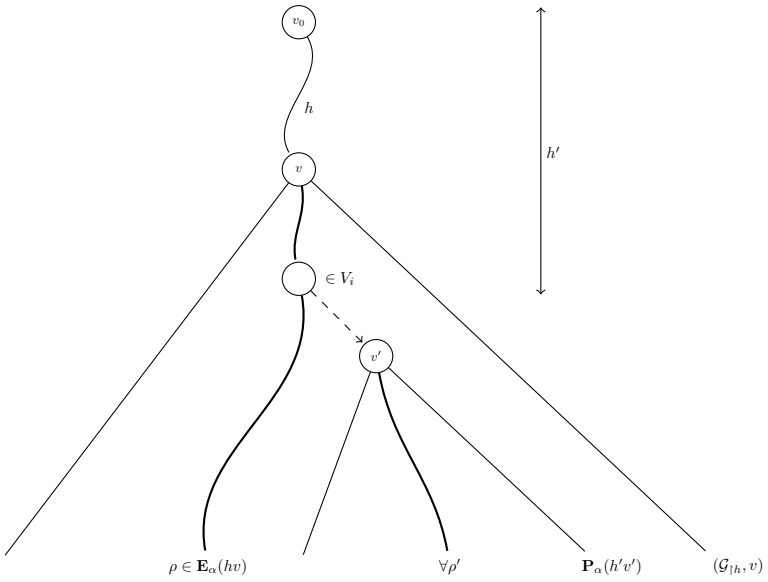


Figure 4 $\rho \in \mathbf{E}_\alpha(hv)$.

Proof. Let us show, by induction on α, that $\langle \bar{\sigma}_{\restriction h} \rangle_v \in \mathbf{P}_\alpha(hv)$ for all $hv \in \mathrm{Hist}(v_0)$.

For $\alpha = 0$, we have $\langle \bar{\sigma}_{\restriction h} \rangle_v \in \mathbf{P}_\alpha(hv)$ by definition of $\mathbf{P}_0(hv)$.

Let $\alpha + 1$ be a successor ordinal. By induction hypothesis, we have that $\langle \bar{\sigma}_{\restriction h} \rangle_v \in \mathbf{P}_\alpha(hv)$ for all $hv \in \mathrm{Hist}(v_0)$. Suppose that there exists hv such that $\langle \bar{\sigma}_{\restriction h} \rangle_v \notin \mathbf{P}_{\alpha+1}(hv)$, i.e. $\langle \bar{\sigma}_{\restriction h} \rangle_v \in \mathbf{E}_\alpha(hv)$. This means that there is a history $h' = hg \in \mathrm{Hist}_i$ for some $i \in \Pi$ with $hv \leq h' < h\rho$, and there exists a vertex v' with $h'v' \not< h\rho$, such that $\forall \rho' \in \mathbf{P}_\alpha(h'v')$, $\lambda_i(h \cdot \langle \bar{\sigma}_{\restriction h} \rangle_v) > \lambda_i(h'\rho')$. In particular, by induction hypothesis

$$\lambda_i(h \cdot \langle \bar{\sigma}_{\restriction h} \rangle_v) > \lambda_i(h' \cdot \langle \bar{\sigma}_{\restriction h'} \rangle_{v'}). \tag{4}$$

Let us consider the player i strategy σ_i' in $(\mathcal{G}_{\restriction h}, v)$ such that $g \cdot \langle \bar{\sigma}_{\restriction h'} \rangle_{v'}$ is consistent with σ_i'. Then σ_i' is a finitely deviating strategy with the (unique) g-deviation step from $\bar{\sigma}_{\restriction h}$. It is a profitable deviation for player i in $(\mathcal{G}_{\restriction h}, v)$ by (4), a contradiction with $\bar{\sigma}$ being a weak SPE.

Let α be a limit ordinal. By induction hypothesis $\langle \bar{\sigma}_{\restriction h} \rangle_v \in \mathbf{P}_\beta(hv), \forall \beta < \alpha$. Therefore $\langle \bar{\sigma}_{\restriction h} \rangle_v \in \mathbf{P}_\alpha(hv) = \bigcap_{\beta<\alpha} \mathbf{P}_\beta(hv)$. ◀

▶ **Lemma 20.** *Suppose that* $\mathbf{P}_{\alpha_*}(hv) \neq \emptyset$ *for all* $hv \in \mathrm{Hist}(v_0)$*, and let* $\rho \in \mathbf{P}_{\alpha_*}(v_0)$*. Then* $(\mathcal{G}, v_0)$ *has a weak SPE with outcome* ρ*.*

Proof. We are going to show how to construct a very weak SPE $\bar{\sigma}$ (and thus a weak SPE by Proposition 13) with outcome ρ. The construction of $\bar{\sigma}$ is done step by step thanks to a progressive labeling of the histories $hv \in \mathrm{Hist}(v_0)$. Let us give an intuitive idea of the construction of $\bar{\sigma}$. Initially, we partially construct $\bar{\sigma}$ such that it produces an outcome in $(\mathcal{G}, v_0)$ equal to $\rho \in \mathbf{P}_{\alpha_*}(v_0)$; we also label each non-empty prefix of ρ by ρ. Then we consider a shortest non-labeled history $h'v'$, and we correctly choose some $\rho' \in \mathbf{P}_{\alpha_*}(h'v')$ (we will see later how). We continue the construction of $\bar{\sigma}$ such that it produces the outcome ρ' in $(\mathcal{G}_{\restriction h'}, v')$, and for each non-empty prefix g of ρ', we label $h'g$ by ρ' (notice that the prefixes of h' have already been labeled by choice of h'). And so on. In this way, the labeling is a

map $\gamma : \mathrm{Hist}(v_0) \to \bigcup_{hv} \mathbf{P}_{\alpha_*}(hv)$ that allows to recover from $h'g$ the outcome ρ' of $\bar{\sigma}_{\restriction h'}$ in $(\mathcal{G}_{\restriction h'}, v')$ of which g is prefix. Let us now go into the details.

Initially, none of the histories is labeled. We start with history v_0 and the given play $\rho \in \mathbf{P}_{\alpha_*}(v_0)$. The strategy profile $\bar{\sigma}$ is partially defined such that $\langle \bar{\sigma} \rangle_{v_0} = \rho$, that is, if $\rho = \rho_0\rho_1 \ldots$, then $\sigma_i(\rho_{\leq n}) = \rho_{n+1}$ for all $\rho_n \in V_i$ and $i \in \Pi$. The non-empty prefixes h of ρ are all labeled with $\gamma(h) = \rho$.

At the following steps, we consider a history $h'v'$ that is not yet labeled, but such that h' has already been labeled. By induction, $\gamma(h') = \langle \bar{\sigma}_{\restriction h} \rangle_v$ such that $hv \leq h'$. Suppose that $\mathrm{Last}(h') \in V_i$, we then choose a play $\rho' \in \mathbf{P}_{\alpha_*}(h'v')$ such that (see Figure 5)

$$\lambda_i(h \cdot \langle \bar{\sigma}_{\restriction h} \rangle_v) \leq \lambda_i(h'\rho'). \tag{5}$$

Such a play ρ' exists for the next reasons. By induction, we know that $\langle \bar{\sigma}_{\restriction h} \rangle_v \in \mathbf{P}_{\alpha_*}(hv)$. Since $\mathbf{P}_{\alpha_*}(hv) = \mathbf{P}_{\alpha_*+1}(hv)$ by Proposition 17, we have $\langle \bar{\sigma}_{\restriction h} \rangle_v \notin \mathbf{E}_{\alpha_*}(hv)$, and we get the existence of ρ' by definition of $\mathbf{E}_{\alpha_*}(hv)$. We continue to construct $\bar{\sigma}$ such that $\langle \bar{\sigma}_{\restriction h'} \rangle_{v'} = \rho'$, i.e. if $\rho' = \rho'_0\rho'_1 \ldots$, then $\sigma_i(h'\rho'_{\leq n}) = h'\rho'_{n+1}$ for all $\rho'_n \in V_i$ and $i \in \Pi$. For all non-empty prefixes g of ρ', we define $\gamma(h'g) = \rho'$ (notice that the prefixes of h' are already labeled).

Let us show that the constructed profile $\bar{\sigma}$ is a very weak SPE. Consider a history $hv \in \mathrm{Hist}_i$ for some $i \in \Pi$, and a one-shot deviating strategy σ'_i from $\bar{\sigma}_{\restriction h}$ in the subgame $(\mathcal{G}_{\restriction h}, v)$. Let v' be such that $\sigma'_i(v) = v'$. By definition of $\bar{\sigma}$, we have $\gamma(hv) = \langle \bar{\sigma}_{\restriction g} \rangle_u$ for some history $gu \leq hv$ and $h \cdot \langle \bar{\sigma}_{\restriction h} \rangle_v = g \cdot \langle \bar{\sigma}_{\restriction g} \rangle_u$; and we have also $\gamma(hvv') = \langle \bar{\sigma}_{\restriction hv} \rangle_{v'}$. Moreover $\lambda_i(g \cdot \langle \bar{\sigma}_{\restriction g} \rangle_u) \leq \lambda_i(hv \cdot \langle \bar{\sigma}_{\restriction hv} \rangle_{v'})$ by (5), and $\lambda_i(hv \cdot \langle \bar{\sigma}_{\restriction hv} \rangle_{v'}) = \lambda_i(h \cdot \langle \sigma'_i, \sigma_{-i\restriction h} \rangle_v)$ because σ'_i is one-shot deviating. Therefore

$$\lambda_i(h \cdot \langle \bar{\sigma}_{\restriction h} \rangle_v) = \lambda_i(g \cdot \langle \bar{\sigma}_{\restriction g} \rangle_u) \leq \lambda_i(hv \cdot \langle \bar{\sigma}_{\restriction hv} \rangle_{v'}) = \lambda_i(h \cdot \langle \sigma'_i, \sigma_{-i\restriction h} \rangle_v)$$

which shows that $\bar{\sigma}_{\restriction h}$ is a very weak NE in $(\mathcal{G}_{\restriction h}, v)$. Hence $\bar{\sigma}$ is a very weak SPE, and thus also a weak SPE. ◀

4 Quantitative Reachability Games

In this section, we focus on quantitative reachability games, such that the player i cost of a play is the number of edges to reach his target set T_i (see Definition 2). Recall that for those games, SPEs, weak SPEs, and very weak SPEs, are equivalent notions (see Corollary 15). It is known that there always exists an SPE in quantitative reachability games [1, 7].

▶ **Theorem 21** ([1, 7]). *Each quantitative reachability game $(\mathcal{G}, v_0)$ has an SPE.*

As SPEs and weak SPEs coincide in quantitative reachability games, we get the next result by Theorem 18.

▶ **Corollary 22.** *Let $(\mathcal{G}, v_0)$ be a quantitative reachability game. The sets $\mathbf{P}_{\alpha_*}(hv)$ are non-empty, for all $hv \in \mathrm{Hist}(v_0)$, and $\mathbf{P}_{\alpha_*}(v_0)$ is the set of outcomes of SPEs in $(\mathcal{G}, v_0)$.*

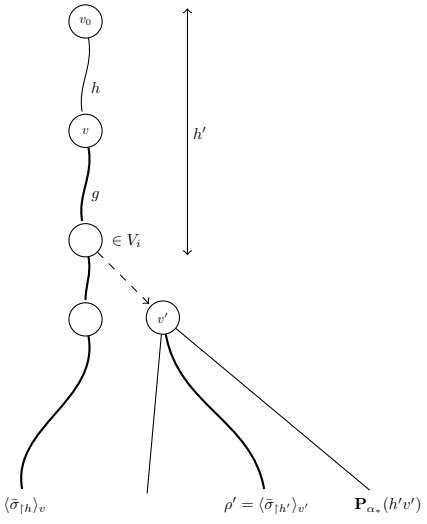


Figure 5 Construction of a very weak SPE $\bar{\sigma}$.

The proof provided for Theorem 21 is non constructive since it relies on topological arguments. Our main result is that one can algorithmically construct an SPE in $(\mathcal{G}, v_0)$ that is moreover finite-memory, thanks to the sets $\mathbf{P}_{\alpha_*}(hv)$.

▶ **Theorem 23.** *Each quantitative reachability initialized game $(\mathcal{G}, v_0)$ has a finite-memory SPE. Moreover there is an algorithm to construct such an SPE.*

We can also decide whether there exists a (finite-memory) SPE such that the cost of its outcome is component-wise bounded by a given constant vector.

▶ **Corollary 24.** *Let $(\mathcal{G}, v_0)$ be a quantitative reachability initialized game, and let $\bar{c} \in \mathbb{N}^{|\Pi|}$ be a given $|\Pi|$-tuple of integers. Then one can decide whether there exists a (finite-memory) SPE $\bar{\sigma}$ such that $\lambda_i(\langle\bar{\sigma}\rangle_{v_0}) \leq c_i$ for all $i \in \Pi$.*

The main ingredients of the proof of Theorem 23 are the next ones. They will be a little detailed in the sequel of this section, as well as the proof of Corollary 24.

1. Given α, the infinite number of sets $\mathbf{P}_\alpha(hv)$ can be replaced by the finite number of sets $\mathbf{P}^I_\alpha(v)$ where I is the set of players that did not reach their target set along history h.
2. The fixpoint of Proposition 17 is reached with some natural number $\alpha_* \in \mathbb{N}$.
3. Each $\mathbf{P}^I_\alpha(v)$ is a non-empty ω-regular set, thus containing a "lasso play" of the form $h{\cdot}g^\omega$.
4. The lasso plays of each $\mathbf{P}^I_\alpha(v)$ allow to construct a finite-memory SPE.

1. Sets $\mathbf{P}^I_\alpha(v)$. Let $(\mathcal{G}, v_0)$ be a quantitative reachability game. Given a history $h = h_0 \dots h_n$ in $(\mathcal{G}, v_0)$, we denote by $I(h)$ the set of players i such that $\forall k$, $0 \leq k \leq n$, we have $h_k \notin T_i$. In other words $I(h)$ is the set of players that did not reach their target set along history h. If h is empty, then $I(h) = \Pi$. One can prove that sets $\mathbf{P}_\alpha(hv)$ only depend on v and $I(h)$, and thus not on h (we do no longer take care of players that have reached their target set along h). We can thus introduce the notations

$$\mathbf{P}^I_\alpha(v) \quad (\text{resp. } \mathbf{E}^I_\alpha(v))$$

in place of $\mathbf{P}_\alpha(hv)$ (resp. $\mathbf{E}_\alpha(hv)$). In particular, $\mathbf{P}^\Pi_\alpha(v_0) = \mathbf{P}_\alpha(v_0)$ and $\mathbf{E}^\Pi_\alpha(v_0) = \mathbf{E}_\alpha(v_0)$. Hence given α, the infinite number of sets $\mathbf{P}_\alpha(hv)$ can be replaced by the finite number of sets $\mathbf{P}^I_\alpha(v)$.

A key result in the proof of Theorem 23 is the following one: given $\mathbf{P}^I_\alpha(v)$ and $i \in I$, if for all $\rho \in \mathbf{P}^I_\alpha(v)$, we have $\lambda_i(\rho) < +\infty$, then there exists c such that for all $\rho \in \mathbf{P}^I_\alpha(v)$, we have $\lambda_i(\rho) \leq c$. The constant c only depends on α, I, v, and i. As a consequence of this result, we have that $\sup\{\lambda_i(\rho) \mid \rho \in \mathbf{P}^I_\alpha(v)\}$ is equal to $\max\{\lambda_i(\rho) \mid \rho \in \mathbf{P}^I_\alpha(v)\}$. We use the next notation for this maximum: given $\mathbf{P}^I_\alpha(v)$, we define $\overline{\Lambda}(\mathbf{P}^I_\alpha(v))$ such that

$$\Lambda_i(\mathbf{P}^I_\alpha(v)) = \begin{cases} -1 & \text{if } i \notin I, \\ \max\{\lambda_i(\rho) \mid \rho \in \mathbf{P}^I_\alpha(v)\} & \text{if } i \in I. \end{cases}$$

In this definition, -1 indicates that player i has already visited his target set T_i, and the max belongs to $\mathbb{N} \cup \{+\infty\}$.

2. Fixpoint with $\alpha_* \in \mathbb{N}$. To explain why the fixpoint (when computing the sets $\mathbf{P}^I_\alpha(v)$, see Proposition 17) is reached in a finite number of steps, we need to adapt previous notation $\overline{\Lambda}(\mathbf{P}^I_\alpha(v))$ to mention the maximum costs for plays in $\mathbf{P}^I_\alpha(v)$ *starting with edge* (v, v'):

$$\Lambda_i(\mathbf{P}^I_\alpha(v), v') = \begin{cases} -1 & \text{if } i \notin I, \\ \max\{\lambda_i(\rho) \mid \rho \in \mathbf{P}^I_\alpha(v) \text{ and } \rho_0\rho_1 = vv'\} & \text{if } i \in I. \end{cases} \tag{6}$$

In this definition, the max is equal to -1 when $\{\lambda_i(\rho) \mid \rho \in \mathbf{P}^I_\alpha(v) \text{ and } \rho_0\rho_1 = vv'\} = \emptyset$. One can prove that if $\mathbf{P}^I_\alpha(v) \neq \mathbf{P}^I_{\alpha+1}(v)$, then there exist $J \subseteq \Pi$ and $(u,u') \in E$ such that $\overline{\Lambda}(\mathbf{P}^J_\alpha(u),u') \neq \overline{\Lambda}(\mathbf{P}^J_{\alpha+1}(u),u')$. Notice that there is a finite number of sequences $(\overline{\Lambda}(\mathbf{P}^I_\alpha(v),v'))_\alpha$ as they only depend on $I \subseteq \Pi$ and $(v,v') \in E$, and that they are nonincreasing for the usual component-wise ordering over $(\mathbb{N} \cup \{-1,+\infty\})^\Pi$. As this ordering is a well-quasi-ordering, there exists an integer (and not only an ordinal) α'_* such that $\overline{\Lambda}(\mathbf{P}^I_{\alpha'_*}(v),v') = \overline{\Lambda}(\mathbf{P}^I_{\alpha'_*+1}(v),v')$ for all $I \subseteq \Pi$ and $(v,v') \in E$. We get that $\alpha_* \leq \alpha'_*$, showing that $\alpha_* \in \mathbb{N}$.

3. The sets $\mathbf{P}^I_\alpha(v)$ are ω-regular. We prefer to show that each set $\mathbf{P}^I_\alpha(v)$ is MSO-definable, instead of providing the (technical) construction of a Büchi automaton, It is well-known that a set of ω-words is ω-regular iff it is MSO-definable, by Büchi theorem [18]. Moreover from the Büchi automaton, one can construct an equivalent MSO-sentence, and conversely. One can also decide whether an MSO-sentence is satisfiable [18]. We recall that MSO-logic uses, in addition to variables $x,y,\ldots$ ($X,Y,\ldots$ resp.) that describe a position (a set of positions resp.) in an ω-word ρ, the relations $x<y$, $Succ(x,y)$, $X(x)$ to mention that $x \in X$, and $Q_u(x)$ to mention that vertex u is at position x of ρ.

As a first step, we show that if $\mathbf{P}^I_\alpha(v)$ is MSO-definable, say by sentence ϕ, then $\overline{\Lambda}(\mathbf{P}^I_\alpha(v))$ is computable. By definition $\Lambda_i(\mathbf{P}^I_\alpha(v))$ equals -1 if $i \notin I$, and is thus computable in this case. Given $i \in I$, one can decide whether $\Lambda_i(\mathbf{P}^I_\alpha(v)) = +\infty$ by checking whether $\phi \wedge \varphi$ is satisfiable, with $\varphi = \forall x \cdot \neg(\vee_{u \in T_i} Q_u(x))$. In case of a positive answer, $\Lambda_i(\mathbf{P}^I_\alpha(v))$ is thus computable. If not, one can prove that $\Lambda_i(\mathbf{P}^I_\alpha(v)) < n$ where n is the number of states of a Büchi automaton accepting $\mathbf{P}^I_\alpha(v)$. We can then similarly test whether $\Lambda_i(\mathbf{P}^I_\alpha(v)) = c$ by considering decreasing constants c from $n-1$ to 0, and thus compute $\Lambda_i(\mathbf{P}^I_\alpha(v))$.

As a second step, we show that each $\mathbf{P}^I_\alpha(v)$ is MSO-definable by induction on α. For $\alpha = 0$, as $\mathbf{P}^I_0(v)$ is the set of plays starting with v, the required sentence is $Q_v(0) \wedge \forall x \cdot \vee_{(u,u') \in E}(Q_u(x) \wedge Q_{u'}(x+1))$. Let $\alpha \in \mathbb{N}$, and suppose that $\mathbf{P}^I_\alpha(v)$ is a fixed MSO-definable set. To show that $\mathbf{P}^I_{\alpha+1}(v)$ is also MSO-definable, it is enough to show that $\mathbf{E}^I_\alpha(v)$ is MSO-definable. Thanks to $\overline{\Lambda}(\mathbf{P}^I_\alpha(v))$, the definition of $\rho \in \mathbf{E}^I_\alpha(v)$ (see Definition 16) can be rephrased as follows: there exist $n \in \mathbb{N}$, $i \in I$, and $u,u',v' \in V$ with $u' \neq v'$, $(u,v') \in E$, such that $\rho_n = u \in V_i$, $\rho_{n+1} = u'$, and $\lambda_i(\rho) > \Lambda_i(\mathbf{P}^{J'}_\alpha(v')) + (n+1)$, where J' is the subset of players of I that did not visit their target set along $\rho_{\leq n}$. Notice that this inequality implies that $i \in J'$ and $\Lambda_i(\mathbf{P}^{J'}_\alpha(v')) < +\infty$. The sentence ψ defining $\mathbf{E}^I_\alpha(v)$ is the following one:

$$\exists n \cdot \bigvee_{\substack{u,u' \neq v' \in V \\ (u,v') \in E}} \bigvee_{J' \subseteq I} \bigvee_{\substack{i \in J', u \in V_i \\ \Lambda_i(\mathbf{P}^{J'}_\alpha(v')) < +\infty}} (Q_u(n) \wedge Q_{u'}(n+1) \wedge \phi_{J',n} \wedge \varphi_{J',n,v',i}) \,.$$

In sentence ψ, we use $\phi_{J',n}$ expressing the definition of J', and $\varphi_{J',n,v',i}$ expressing that if player i visits its target set along ρ, it is after $\Lambda_i(\mathbf{P}^{J'}_\alpha(v')) + n + 1$ edges from ρ_0. Notice that the (computable) constant $\Lambda_i(\mathbf{P}^{J'}_\alpha(v'))$ can be considered as fixed, since $\mathbf{P}^I_\alpha(v)$ is fixed.

As a third step, as each $\mathbf{P}^I_\alpha(v)$ is ω-regular, then for all $i \in I$, $\mathbf{P}^I_{\alpha_*}(v)$ has a computable lasso play $\rho = h_{i,I,v}\cdot(g_{i,I,v})^\omega$ (depending on i, I, and v) with maximal cost $\lambda_i(\rho) = \Lambda_i(\mathbf{P}^I_{\alpha_*}(v))$.

4. Construction of a finite-memory SPE. We have all the ingredients to prove that each quantitative reachability game has a computable finite-memory SPE. The procedure is the same as the one developed in the proof of Lemma 20, except that it uses the lasso plays $h_{i,I,v}\cdot(g_{i,I,v})^\omega \in \mathbf{P}^I_{\alpha_*}(v)$. For the initial history v_0, we use any play $h_{i,\Pi,v_0}\cdot(g_{i,\Pi,v_0})^\omega \in \mathbf{P}^\Pi_{\alpha_*}(v_0)$, $i \in \Pi$. At the following steps, given a not yet labeled history $h'v'$, the proof of Lemma 20 requires to choose a play $\rho' \in \mathbf{P}^{J'}_{\alpha_*}(v')$ (for a certain $J' \subseteq I$) with a cost $\lambda_i(h'\rho')$ sufficiently

large. We simply choose $\rho' = h_{i,J',v'} \cdot (g_{i,J',v'})^\omega$ that has maximal cost $\lambda_i(\rho') = \Lambda_i(\mathbf{P}^{J'}_{\alpha_*}(v'))$. This strategy profile $\bar{\sigma}$ is an SPE that is finite-memory since it depends on the finite number of lasso plays $h_{j,I,v} \cdot (g_{j,I,v})^\omega$.

It remains to prove the decidability of the constrained existence of an SPE for quantitative reachability games, as announced in Corollary 24. Let $\bar{c} \in \mathbb{N}^{|\Pi|}$ be a constant vector. We know that the set $\mathbf{P}^{\Pi}_{\alpha_*}(v_0)$ of outcomes of SPEs in $(\mathcal{G}, v_0)$ is MSO-definable. It is easy to see that the set $\{\rho \mid \rho \in \mathbf{P}^{\Pi}_{\alpha_*}(v_0) \text{ and } \lambda_i(\rho) \leq c_i, \forall i\}$ is also MSO-definable. We can thus decide whether this set is non-empty, which means that the constrained existence of an SPE is decidable. In case of positive answer, this set contains a lasso play $h \cdot g^\omega$. Exactly as done above, one can construct a finite-memory SPE $\bar{\sigma}$ such that $\langle \bar{\sigma} \rangle_{v_0} = h \cdot g^\omega$.

To conclude this section, we present another large class of games for which one can decide whether there exists a weak SPE.[8] The hypotheses are general conditions on the cost functions λ_i, $i \in \Pi$:

▶ **Theorem 25.** *Let $(\mathcal{G}, v_0)$ be an initialized game such that:*

- *each cost function λ_i is prefix-independent, and with finite range $C_i \subset \mathbb{Q}$,*
- *for all $i \in \Pi$, $c_i \in C_i$, and $v \in V$, the set of plays ρ in $(\mathcal{G}, v)$ with $\lambda_i(\rho) = c_i$ is an ω-regular set.*

Then one can decide whether $(\mathcal{G}, v_0)$ has a weak SPE $\bar{\sigma}$ (resp. such that $\lambda_i(\langle \bar{\sigma} \rangle_{v_0}) \leq c_i$ forall i for given $c_i \in C_i$, $i \in \Pi$). In case of positive answer, one can construct such a finite-memory weak SPE.

For example, the hypotheses of this theorem are satisfied by the liminf games and the limsup games; they are also satisfied by the game of Example 12. The proof of this theorem shares similar points with the proof given for quantitative reachability games. Again, it uses our Folk Theorem for weak SPEs.

5 Conclusion and Future Work

In this article, we have studied the existence of (weak) SPEs in quantitative games. We have proposed a Folk Theorem that characterizes all the outcomes of weak SPEs. To illustrate the potential of this theorem, we have given two applications. The first one is concerned with quantitative reachability games for which we have provided an algorithm to compute a finite-memory SPE, and a second algorithm for deciding the constrained existence of a (finite-memory) SPE. The second application is concerned with another large class of games for which we have proved that the (constrained) existence of a (finite-memory) weak SPE is decidable.

Future possible directions of research are the following ones. We would like to study the complexities of the problems studied for the two classes of games. We also want to investigate the application of our Folk Theorem to other classes of games. The example of Figure 3 is a game with a weak SPE but no SPE (see Example 12). Recall that for this game, the cost $\lambda_i(\rho)$ can be seen as either the mean-payoff, or the liminf, or the limsup, of the weights of ρ. We do not know if games with this kind of payoff functions always have a weak SPE or not. Notice that using a variant of the techniques developed for weak SPEs, we were able to obtain a Folk theorem for SPEs, nevertheless with a weaker characterization since one implication requires that the cost functions are upper-semicontinuous (see [2]).

[8] Contrarily to quantitative reachability games, we do not know if a weak SPE always exists for games in this class.

References

1 Thomas Brihaye, Véronique Bruyère, Julie De Pril, and Hugo Gimbert. On subgame perfection in quantitative reachability games. *Logical Methods in Computer Science*, 9, 2012.

2 Thomas Brihaye, Véronique Bruyère, Noémie Meunier, and Jean-François Raskin. Weak subgame perfect equilibria and their application to quantitative reachability. *CoRR*, abs/1504.01557, 2015.

3 Thomas Brihaye, Julie De Pril, and Sven Schewe. Multiplayer cost games with simple Nash equilibria. In *LFCS*, volume 7734 of *Lecture Notes in Comput. Sci.*, pages 59–73. Springer, 2013.

4 Véronique Bruyère, Noémie Meunier, and Jean-François Raskin. Secure equilibria in weighted games. In *CSL-LICS*, pages 26:1–26:26. ACM, 2014.

5 Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Quantitative languages. *ACM Trans. Comput. Log.*, 11, 2010.

6 János Flesch, Jeroen Kuipers, Ayala Mashiah-Yaakovi, Gijs Schoenmakers, Eilon Solan, and Koos Vrieze. Perfect-information games with lower-semicontinuous payoffs. *Math. Oper. Res.*, 35:742–755, 2010.

7 Drew Fudenberg and David Levine. Subgame-perfect equilibria of finite- and infinite-horizon games. *Journal of Economic Theory*, 31:251–268, 1983.

8 Drew Fudenberg and Jean Tirole. *Game Theory*. MIT Press, 1991.

9 Miroslav Klimos, Kim G. Larsen, Filip Stefanak, and Jeppe Thaarup. Nash equilibria in concurrent priced games. In *LATA*, volume 7183 of *Lecture Notes in Comput. Sci.*, pages 363–376. Springer, 2012.

10 H.W. Kuhn. Extensive games and the problem of information. *Classics in Game Theory*, pages 46–68, 1953.

11 Jean-François Mertens. Repeated games. In *Internat. Congress Mathematicians*, American Mathematical Society, pages 1528–1577. Providence, RI, 1987.

12 J. F. Nash. Equilibrium points in n-person games. In *PNAS*, volume 36, pages 48–49. National Academy of Sciences, 1950.

13 Martin J. Osborne and Ariel Rubinstein. *A course in Game Theory*. MIT Press, Cambridge, MA, 1994.

14 Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *POPL*, pages 179–190. ACM Press, 1989.

15 Stéphane Le Roux and Arno Pauly. Infinite sequential games with real-valued payoffs. In *CSL-LICS*, pages 62:1–62:10. ACM, 2014.

16 Ariel Rubinstein. Comments on the interpretation of game theory. *Econometrica*, 59:909–924, 1991.

17 Eilon Solan and Nicolas Vieille. Deterministic multi-player dynkin games. *Journal of Mathematical Economics*, 39:911–929, 2003.

18 Wolfgang Thomas. Automata on infinite objects. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 133–192. Elsevier Science Publishers, Amsterdam, 1990.

19 Michael Ummels. Rational behaviour and strategy construction in infinite multiplayer games. In *FSTTCS*, volume 4337 of *Lecture Notes in Comput. Sci.*, pages 212–223. Springer, 2006.

What are Strategies in Delay Games? Borel Determinacy for Games with Lookahead*

Felix Klein and Martin Zimmermann

Reactive Systems Group, Saarland University, 66123 Saarbrücken, Germany
{klein, zimmermann}@react.uni-saarland.de

Abstract

We investigate determinacy of delay games with Borel winning conditions, infinite-duration two-player games in which one player may delay her moves to obtain a lookahead on her opponent's moves.

First, we prove determinacy of such games with respect to a fixed evolution of the lookahead. However, strategies in such games may depend on information about the evolution. Thus, we introduce different notions of universal strategies for both players, which are evolution-independent, and determine the exact amount of information a universal strategy needs about the history of a play and the evolution of the lookahead to be winning. In particular, we show that delay games with Borel winning conditions are determined with respect to universal strategies. Finally, we consider decidability problems, e.g., "Does a player have a universal winning strategy for delay games with a given winning condition?", for ω-regular and ω-context-free winning conditions.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases Determinacy, Infinite Games, Delay Games, Borel Hierarchy

Digital Object Identifier 10.4230/LIPIcs.CSL.2015.519

1 Introduction

Determinacy is the most fundamental property of a game: a game is determined, if one of the players has a winning strategy. One can even argue that a determinacy result paved the way for game theory: in 1913, Zermelo proved what is today known as Zermelo's theorem [18]: every two-player zero-sum game of perfect information and finite duration is determined.

In this work, we are concerned with the infinite-duration variant of such games, so-called Gale-Stewart games. Such a game is played between Player I and Player O in rounds $i \in \mathbb{N}$: in round i, Player I picks a letter $\alpha(i) \in \Sigma_I$ and then Player O picks a letter $\beta(i) \in \Sigma_O$. Player O wins, if the outcome $\binom{\alpha(0)}{\beta(0)}\binom{\alpha(1)}{\beta(1)}\binom{\alpha(2)}{\beta(2)}\cdots$ is in the winning condition $L \subseteq (\Sigma_I \times \Sigma_O)^\omega$. Accordingly, a strategy for Player I is a function $\tau\colon \Sigma_O^* \to \Sigma_I$ mapping the previous moves of Player O to the next letter from Σ_I to be picked. The definition for Player O is dual. Note that a strategy cannot access the previous moves determined by itself. This is not a restriction, as they can always be reconstructed.

Let $\rho(\tau, \sigma)$ denote the outcome of the play where Player I employs the strategy τ and Player O the strategy σ. Then, determinacy can be characterized as follows: the negation $\forall\sigma\exists\tau.\, \rho(\tau,\sigma) \notin L$ of $\exists\sigma\forall\tau.\, \rho(\tau,\sigma) \in L$ is equivalent to $\exists\tau\forall\sigma.\, \rho(\tau,\sigma) \notin L$, i.e., the order of the quantifiers can be swapped.

* Partially supported by the projects "TriCS" (ZI 1516/1-1) and "AVACS" (SFB/TR 14) of the German Research Foundation (DFG). The first author was supported by an IMPRS-CS PhD Scholarship.

24th EACSL Annual Conference on Computer Science Logic (CSL 2015).
Editor: Stephan Kreutzer; pp. 519–533

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Gale-Stewart games are an important tool in set theory and a long line of research into determinacy results for such games culminated in Martin's seminal Borel determinacy theorem [11]: every Gale-Stewart game with a Borel winning condition is determined. On the other hand, using the axiom of choice, one can construct non-determined games. Even more so, determinacy of games with ω-context-free conditions, which are not necessarily Borel, is equivalent to a large cardinal assumption that is not provable in ZFC [5].

Gale-Stewart games also have important applications in theoretical computer science as they subsume games studied in automata theory, e.g., parity games and LTL realizability games, and constitute the foundation of game-based synthesis, the solution to Church's problem [1]. Showing the winning condition of a game to be Borel and then applying Martin's theorem is typically the simplest proof of determinacy for a novel winning condition. However, one can typically obtain stronger results, e.g., positional determinacy for parity games [3, 14]. The quantifier swap induced by this determinacy result underlies (implicitly or explicitly) all complementation proofs for parity tree automata, the crucial step in proving decidability of monadic second-order logic over infinite trees.

Delay Games. Oftentimes, the strict alternation of moves in a Gale-Stewart game is too restrictive to model applications in computer science, e.g., in the presence of asynchronous components, buffers, or communication between components. Delay games, a relaxation of Gale-Stewart games, model such situations by allowing Player O to delay her moves in order to obtain a lookahead on her opponent's moves. This gives her an advantage and allows her to win games she would lose without lookahead.

Furthermore, delay games have deep connections to uniformization problems for relations w.r.t. continuous functions [15, 16]. Consider a winning condition $L \subseteq (\Sigma_I \times \Sigma_O)^\omega$: a winning strategy σ for Player O in a game with winning condition L induces a mapping $\lambda_\sigma \colon \Sigma_I^\omega \to \Sigma_O^\omega$ such that $\{\binom{\alpha}{\lambda_\sigma(\alpha)} \mid \alpha \in \Sigma_I^\omega\} \subseteq L$: we say that λ_σ uniformizes L. If σ is winning for the Gale-Stewart game with winning condition L, then λ_σ is causal: the n-th letter of $\lambda_\sigma(\alpha)$ only depends on the first n letters of α. Furthermore, if σ is winning in the delay game with winning condition L then λ_σ is continuous in the Cantor topology. The latter result can even be refined: if σ only delays moves a bounded number of times during each play, then λ_σ is Lipschitz-continuous. Thus, uniformization problems w.r.t. (Lipschitz-)continuous functions are reducible to solving delay games.

To capture and to analyze the precise amount of lookahead that is necessary to win, delay games are defined w.r.t. so-called delay functions, which represent the evolution of the lookahead. Thus, formally Player O does not decide to skip a move, but the delay function determines how many moves she skips: given a delay function $f \colon \mathbb{N} \to \mathbb{N}_+$, the delay game $\Gamma_f(L)$ is played in rounds, where in round i Player I has to pick $f(i)$ letters and afterwards Player O has to pick a single letter. Thus, if $f(i) > 1$, then Player O's lookahead increases by $f(i) - 1$ letters. Typically, one is interested in the existence of a delay function f that allows Player O to win $\Gamma_f(L)$. One could imagine an alternative formalization where Player O may explicitly skip moves at her own choice. We will encounter this variant in Section 5, where it is shown to be *equivalent* to the one using delay functions.

Delay games where introduced by Hosch and Landweber who proved decidability of the existence of winning strategies with bounded lookahead for games with ω-regular winning conditions [8]. Later, Holtmann, Kaiser, and Thomas [7] proved that for such winning conditions, Player O has a winning strategy with bounded lookahead if and only if she has one with arbitrary lookahead, i.e., bounded lookahead always suffices for ω-regular winning conditions. Furthermore, they gave a doubly-exponential upper bound on the necessary

lookahead and a solution algorithm with doubly-exponential running time. These results were recently improved [10] by showing a tight exponential bound on the necessary lookahead and ExpTime-completeness of solving delay games with ω-regular winning conditions. Finally, delay games with deterministic ω-context-free winning conditions are undecidable [6], while games with max-regular winning conditions w.r.t. bounded lookahead are decidable [19]. All these results can be expressed in terms of uniformization as well.

For all types of winning conditions mentioned above, delay games w.r.t. a fixed delay function are determined [6, 7, 10, 19]: these results are all ad-hoc as they rely on the existence of a deterministic automaton recognizing the winning condition and on determinacy of parity games on countable arenas: one can model the delay game as such a parity game where each vertex contains the whole history of the play as well as the state the automaton reaches when processing this history.

What are Strategies in Delay Games? The most important aspect of a game are its (winning) strategies, e.g., in controller synthesis it is a winning strategy for the player representing the system that is turned into a controller.

In a delay game, the notion of strategy is more complex than in a Gale-Stewart game due to the existence of the delay function: a strategy for Player I is of the form $\tau\colon \Sigma_O^* \to \Sigma_I^*$ with $|\tau(\beta(0)\cdots\beta(i-1))| = f(i)$, as he has to determine $f(i)$ letters in round i. Thus, a strategy for Player I syntactically depends on f and both players' strategies may depend semantically on f. On the one hand, this means that a winning strategy for a game w.r.t. a delay function f might not be applicable for an $f' \neq f$. On the other hand, dependence on a particular delay function enables the reconstruction of the own previous moves.

However, the classical definition of strategies for delay games introduced above is not useful when it comes to applications in synthesis: the lack of robustness with regard to changes in the delay function is a serious problem. Furthermore, determinacy for delay games w.r.t. fixed delay functions is a rather unsatisfactory statement: for every f, either Player I has a winning strategy for $\Gamma_f(L)$ or Player O has one. If $\rho(f,\tau,\sigma)$ denotes the outcome resulting from Player I employing τ and Player O employing σ in a game w.r.t. f, then the negation of $\exists\sigma\forall\tau.\,\rho(f,\tau,\sigma) \in L$ is equivalent to $\exists\tau\forall\sigma.\,\rho(f,\tau,\sigma) \notin L$. However, the function f is quantified *outside* of the negation.

Pushing the negation over the quantification of f yields a much stronger statement, e.g., either there is an f such that Player O wins $\Gamma_f(L)$ or Player I has a strategy that wins $\Gamma_f(L)$ w.r.t. every f. Note that such a strategy has to be universally applicable and winning for every $\Gamma_f(L)$ and may therefore neither syntactically nor semantically depend on a fix delay function. Thus, a determinacy result w.r.t. such universal strategies means that the negation of $\exists f\exists\sigma\forall\tau.\,\rho(f,\tau,\sigma) \in L$ is equivalent to $\exists\tau\forall f\forall\sigma.\,\rho(f,\tau,\sigma) \notin L$, which is arguably a more natural notion.

Our Contribution. We study determinacy results for delay games with and without respect to fixed delay functions and with Borel winning conditions.

Firstly, for games with fixed delay functions, we show determinacy w.r.t. classical strategies that may depend on the function under consideration. This result generalizes all previous determinacy results obtained via reductions to countable parity games using deterministic automata recognizing the winning condition [6, 7, 10, 19].

Secondly, we introduce universal strategies for delay games: for Player I, we consider four variants that differ in the amount of information about a play's history they can access: the previous moves made by the strategy (which are not necessarily reconstructible) and

the evolution of the lookahead in the previous rounds. We compare the strength of these strategies in terms of games they are able to win and show that they form a hierarchy whose first three levels are strict and that strategies in the fourth level are sufficient to win every game that is winable. It is open whether the inclusion between the last two levels is strict or not. For Player O, we only consider two notions of universal strategies, as the second one is already sufficient to win every winable game. Furthermore, we show that the hierarchy for Player O is strict, too.

Thirdly, we consider decision problems of the form "Does a player have a universal strategy for games with some given winning condition L?" for ω-regular and ω-context-free winning conditions. We prove decidability (and tight complexity results) for both players in the ω-regular case and for Player O in the deterministic ω-context-free case. The other case and both cases for non-deterministic ω-context-free winning conditions are undecidable.

This work is meant as a starting point into the investigation of more general notions of strategies in delay games that are independent of the exact evolution of the lookahead and into determinacy results w.r.t. these notions of strategies. We raise many open problems that are left open here. Most importantly, the exact amount of information about a play's history that is necessary to implement a universal strategy for Player I is open. Also, most of the decision problems remain open for the weaker notions of universal strategies we introduce. Finally, we expect there to be other natural notions of universal strategies for delay games, which might not have to be winning for every delay function (a very strong requirement), but for all f for which the given player wins the delay game $\Gamma_f(L)$ using classical strategies.

2 Preliminaries

The set of non-negative integers is denoted by $\mathbb{N}$ and we define $\mathbb{N}_+ = \mathbb{N} \setminus \{0\}$. An alphabet Σ is a non-empty finite set, Σ^* (Σ^n, Σ^ω) denotes the set of finite words (words of length n, infinite words) over Σ. The empty word is denoted by ε and the length of a finite word w by $|w|$. For $w \in \Sigma^* \cup \Sigma^\omega$ and $n \in \mathbb{N}$ we write $w(n)$ for the n-th letter of w.

Delay Games. A delay function is a mapping $f\colon \mathbb{N} \to \mathbb{N}_+$. Given an ω-language $L \subseteq (\Sigma_I \times \Sigma_O)^\omega$ and a delay function f, the game $\Gamma_f(L)$ is played by two players[1], the input player "Player I" and the output player "Player O" in rounds $i \in \mathbb{N}$ as follows: in round i, Player I picks a word $u_i \in \Sigma_I^{f(i)}$, then Player O picks one letter $v_i \in \Sigma_O$. We refer to the sequence $(u_0, v_0)(u_1, v_1)(u_2, v_2)\cdots$ as a play of $\Gamma_f(L)$, which yields two infinite words $\alpha = u_0u_1u_2\cdots$ and $\beta = v_0v_1v_2\cdots$. Player O wins the play if the outcome $\binom{\alpha(0)}{\beta(0)}\binom{\alpha(1)}{\beta(1)}\binom{\alpha(2)}{\beta(2)}\cdots$ is in L, otherwise Player I wins.

Given a delay function f, a strategy for Player I is a mapping $\tau\colon \Sigma_O^* \to \Sigma_I^*$ where $|\tau(w)| = f(|w|)$, and a strategy for Player O is a mapping $\sigma\colon \Sigma_I^* \to \Sigma_O$. Consider a play $(u_0, v_0)(u_1, v_1)(u_2, v_2)\cdots$ of $\Gamma_f(L)$. It is consistent with τ, if $u_i = \tau(v_0 \cdots v_{i-1})$ for every $i \in \mathbb{N}$. It is consistent with σ, if $v_i = \sigma(u_0 \cdots u_i)$ for every $i \in \mathbb{N}$.

▶ Remark. As usual, a strategy has only access to the opponents's moves, but not its own ones. However, this is not a restriction, since they can be reconstructed.

Fix a strategy τ for Player I: in round i, the input to τ is the concatenation $v_0 \cdots v_{i-1}$ of Player O's moves in the previous rounds. The moves $u_0, \ldots, u_{i-1}$ by Player I in the previous rounds are given by $u_j = \tau(v_0 \cdots v_{j-1})$ for every $j < i$.

[1] For pronomial convenience [13], we assume Player I to be male and Player O to be female.

Now, fix a strategy σ for Player O: in round i, the input to σ is the concatenation $u_0 \cdots u_i$ of Player I's moves in the previous rounds, where each u_j for $j \leq i$ satisfies $|u_j| = f(j)$. Thus, the moves $v_0, \ldots, v_{i-1}$ by Player O in the previous rounds are given by $v_j = \sigma(u_0 \cdots u_j)$. Note this construction depends on knowledge about the delay function f, as we decompose the input to σ to obtain the prefix of length $\sum_{j'=0}^{j} f(j')$.

A strategy τ for Player $p \in \{I, O\}$ is winning, if every play that is consistent with τ is winning for Player p. We say that a player wins $\Gamma_f(L)$, if she has a winning strategy and a delay game is determined, if one of the players wins it.

▶ **Example 1.** Consider L_0 over $\{a, b, c\} \times \{b, c\}$ with $\binom{\alpha(0)}{\beta(0)}\binom{\alpha(1)}{\beta(1)}\binom{\alpha(2)}{\beta(2)} \cdots \in L_0$ if $\alpha(n) = a$ for every $n \in \mathbb{N}$ or if $\beta(0) = \alpha(n)$, where n is the smallest position with $\alpha(n) \neq a$. Intuitively, Player O wins, if the letter she picks in the first round is equal to the first letter other than a that Player I picks. Also, Player O wins, if there is no such letter.

We claim that Player I wins $\Gamma_f(L_0)$ for every delay function f: he picks $a^{f(0)}$ in the first round and assume Player O picks b afterwards (the case where she picks c is dual). Then, Player I picks a word starting with c in the second round. The resulting play is winning for Player I no matter how it is continued. Thus, Player I has a winning strategy in $\Gamma_f(L_0)$.

Finally, we also consider delay-free games. Formally, these can be seen as delay games w.r.t. the delay function f with $f(i) = 1$ for every i, i.e., both players pick a single letter in each round. As f is irrelevant, we denote such a game with winning condition L by $\Gamma(L)$.

The Borel Hierarchy. Fix an alphabet Σ. The Borel hierarchy of ω-languages over Σ consists of levels $\mathbf{\Sigma}_\alpha$ and $\mathbf{\Pi}_\alpha$ for every countable ordinal $\alpha > 0$, which are defined inductively by

- $\mathbf{\Sigma}_1 = \{L \subseteq \Sigma^\omega \mid L = K \cdot \Sigma^\omega \text{ for some } K \subseteq \Sigma^*\}$,
- $\mathbf{\Pi}_\alpha = \{\Sigma^\omega \setminus L \mid L \in \mathbf{\Sigma}_\alpha\}$ for every α, and
- $\mathbf{\Sigma}_\alpha = \{\bigcup_{i \in \mathbb{N}} L_i \mid L_i \in \mathbf{\Pi}_{\alpha_i} \text{ with } \alpha_i < \alpha \text{ for every } i\}$ for every $\alpha > 1$.

The following basic properties will be useful later on.

▶ Remark. Let α be a countable ordinal.

- $\mathbf{\Sigma}_\alpha \cup \mathbf{\Pi}_\alpha \subseteq \mathbf{\Sigma}_{\alpha+1} \cap \mathbf{\Pi}_{\alpha+1}$.
- $\mathbf{\Sigma}_\alpha$ and $\mathbf{\Pi}_\alpha$ are closed under finite unions and finite intersections.

A language L is Borel, if it is in one of the levels constituting the Borel hierarchy.

▶ **Theorem 2** (Borel Determinacy Theorem [11]). *Let L be Borel. Then, $\Gamma(L)$ is determined.*

3 Borel Determinacy of Delay Games w.r.t. Fixed Delay Functions

Fix alphabets Σ_I and Σ_O and a fresh skip symbol $\triangleright \notin \Sigma_O$, and define $\Sigma_O^\triangleright = \Sigma_O \cup \{\triangleright\}$. To simplify our notation, let h be the morphism that removes the skip symbol, i.e., the one defined by $h(\triangleright) = \varepsilon$ and $h(a) = a$ for every $a \in \Sigma_O$. Also, given two infinite words α and β we write $\binom{\alpha}{\beta}$ for the word $\binom{\alpha(0)}{\beta(0)}\binom{\alpha(1)}{\beta(1)}\binom{\alpha(2)}{\beta(2)} \cdots$. Analogously, we write $\binom{x}{y}$ for finite words x and y, provided they are of equal length.

Given a delay function f and an infinite word $\beta \in \Sigma_O^\omega$ we define $\mathrm{shift}_f(\beta) \in (\Sigma_O^\triangleright)^\omega$ by

$$\mathrm{shift}_f(\beta) = \triangleright^{f(0)-1}\beta(0)\triangleright^{f(1)-1}\beta(1)\triangleright^{f(2)-1}\beta(2) \cdots .$$

We lift this definition to languages $L \subseteq (\Sigma_I \times \Sigma_O)^\omega$ via $\mathrm{shift}_f(L) = \{\binom{\alpha}{\mathrm{shift}_f(\beta)} \mid \binom{\alpha}{\beta} \in L\}$. Intuitively, $\mathrm{shift}_f(L)$ encodes the delay function f explicitly by postponing Player O's moves

using skip symbols. Thus, the delay game $\Gamma_f(L)$ and the delay-free game $\Gamma(\text{shift}_f(L))$ are essentially equivalent: a winning strategy for Player $p \in \{I, O\}$ in $\Gamma_f(L)$ can directly be translated into a winning strategy for her in $\Gamma(\text{shift}_f(L))$ and vice versa.

The main result of this section states that delay games with Borel winning conditions and w.r.t. fixed delay functions are determined.

▶ **Theorem 3.** *Let L be Borel and let f be a delay function. Then, $\Gamma_f(L)$ is determined.*

Proof. We show that $\text{shift}_f(L)$ is Borel. Then, our claim follows from the Borel determinacy theorem, as $\Gamma(\text{shift}_f(L))$ and $\Gamma_f(L)$ are essentially the same game.

We will prove the following statement, which is not the tightest result provable, but which suffices for our purposes: if $L \subseteq (\Sigma_I \times \Sigma_O)^\omega$ is in $\mathbf{\Sigma}_\alpha$ (in $\mathbf{\Pi}_\alpha$), then $\text{shift}_f(L)$ is in $\mathbf{\Sigma}_{\alpha+2}$ (in $\mathbf{\Pi}_{\alpha+2}$). To this end, the language $U_f = \text{shift}_f((\Sigma_I \times \Sigma_O)^\omega)$ will be useful. Note that U_f contains exactly those plays during which the non-skip symbols are played at the positions consistent with f. It is straightforward to show that U_f is in $\mathbf{\Pi}_2$.

First, assume we have $L \in \mathbf{\Sigma}_1$, i.e., $L = K \cdot (\Sigma_I \times \Sigma_O)^\omega$ for some $K \subseteq (\Sigma_I \times \Sigma_O)^*$. Then, we have $\text{shift}_f(L) = K' \cdot (\Sigma_I \times \Sigma_O^{\triangleright})^\omega \cap U_f$ where K' is equal to

$$\bigcup_{\binom{\alpha(0)}{\beta(0)}\cdots\binom{\alpha(k)}{\beta(k)} \in K} \left\{ \binom{x}{y} \mid y = \triangleright^{f(0)-1}\beta(0)\cdots\triangleright^{f(k)-1}\beta(k) \text{ and } x \in \alpha(0)\cdots\alpha(k)\cdot\Sigma_I^{|y|-(k+1)} \right\},$$

i.e., $\text{shift}_f(L)$ is in $\mathbf{\Pi}_2 \subseteq \mathbf{\Sigma}_3$.

Now, let L be in $\mathbf{\Pi}_\alpha$, i.e., $L = (\Sigma_I \times \Sigma_O)^\omega \setminus L'$ for some $L' \in \mathbf{\Sigma}_\alpha$. Applying the induction hypothesis yields that $\text{shift}_f(L')$ is in $\mathbf{\Sigma}_{\alpha+2}$. We have

$$\text{shift}_f(L) = ((\Sigma_I \times \Sigma_O^{\triangleright})^\omega \setminus \text{shift}_f(L')) \cap U_f,$$

i.e., $\text{shift}_f(L) \in \mathbf{\Pi}_{\alpha+2}$.

Finally, assume we have $L \in \mathbf{\Sigma}_\alpha$ for some $\alpha > 1$, i.e., $L = \bigcup_{i\in\mathbb{N}} L_i$ with $L_i \in \mathbf{\Pi}_{\alpha_i}$ for some $\alpha_i < \alpha$. An application of the induction hypothesis shows that every $\text{shift}_f(L_i)$ is in $\mathbf{\Pi}_{\alpha_i+2}$. Thus, $\text{shift}_f(L) = \bigcup_{i\in\mathbb{N}} \text{shift}_f(L_i)$ is in $\mathbf{\Sigma}_{\alpha+2}$ as $\alpha_i + 2 < \alpha + 2$ for every i. ◀

Furthermore, from the equivalence of $\Gamma(\text{shift}_f(L))$ and $\Gamma_f(L)$, which holds for arbitrary L, we obtain a result that is applicable to non-Borel winning conditions as well.

▶ **Corollary 4.** *If $\Gamma(\text{shift}_f(L))$ is determined, then so is $\Gamma_f(L)$.*

4 Omnipotent Strategies in Delay Games

In this section, we discuss different notions of strategies for delay games. The one introduced in Section 2 is the classical one that was used in previous works [6, 7, 10, 19]. However, such strategies depend on a fixed delay function f, i.e., they are not useful for a game w.r.t. a delay function $f' \neq f$. This is a syntactic dependence in the case of Player I, as he has to determine $f(i)$ letters in round i. But even Player O's strategies may depend implicitly on knowledge about the delay function under consideration, as we will see below.

In this section, we consider several stronger notions of universal strategies, i.e., strategies that are independent of the delay function under consideration. Informally, for Player I such a strategy returns an infinite word $w \in \Sigma_I^\omega$ and the first $f(i)$ letters of w are used in round i of a delay game w.r.t. f. For Player O, a universal strategy still returns a single letter, but it may no longer depend on information about the delay function under consideration. We say that a universal strategy is omnipotent for a winning condition L, if the strategy is winning for every delay game $\Gamma_f(L)$, independently of the choice of f.

▶ **Example 5.** Again, consider the winning condition L_0 from Example 1 and the strategy $\tau\colon \Sigma_O^* \to \Sigma_I^\omega$ given by $\tau(\varepsilon) = a^\omega$, $\tau(bx) = c^\omega$, and $\tau(cx) = b^\omega$ for $x \in \Sigma_O^*$. Intuitively, in round 0, Player I can pick as many a's as f requires and then always picks c (or b), if Player O has picked b (or c) in round 0. This strategy is winning for him w.r.t. every f and therefore omnipotent for L_0.

4.1 Omnipotent Strategies for Player I

We consider the following variants of universal strategies for Player I, which differ in the amount of information about the play's history they base their decision on. In Example 5, the strategy τ only has access to Player O's moves, which is sufficient to be winning. More powerful notions have (directly or indirectly) access to Player I's moves or to information about the delay function under consideration. Note that Player I cannot reconstruct his moves, if he only has access to Player O's moves, but not the delay function under consideration, which explains the need to access his own moves.

1. An *output-tracking* (o.t.) strategy is a map $\tau\colon \Sigma_O^* \to \Sigma_I^\omega$. Let $(u_0, v_0)(u_1, v_1)(u_2, v_2)\cdots$ be a play of $\Gamma_f(L)$ for some f: it is consistent with τ, if u_i is the prefix of length $f(i)$ of $\tau(v_0 \cdots v_{i-1})$. An o.t. strategy bases its decisions only on the moves v_j of Player O for $j \leq i$ and can deduce the number of rounds already played, but has no way to reconstruct Player I's previous moves. In fact, it cannot even reconstruct the number of letters picked by Player I thus far.
2. A *lookahead-counting* (l.c.) strategy is a mapping $\tau\colon \Sigma_O^* \times \mathbb{N} \to \Sigma_I^\omega$. This time, we say that a play $(u_0, v_0)(u_1, v_1)(u_2, v_2)\cdots$ of $\Gamma_f(L)$ for some f is consistent with τ, if u_i is the prefix of length $f(i)$ of $\tau(v_0 \cdots v_{i-1}, \sum_{j=0}^{i-1} f(j))$. A l.c. strategy has access to the opponent's moves and the number of letters picked by Player I thus far. However, this still does not suffice for Player I to reconstruct the actual letters already picked.
3. An *input-output-tracking* (i.o.t.) strategy is a mapping $\tau\colon \Sigma_O^* \times \Sigma_I^* \to \Sigma_I^\omega$. We define a play $(u_0, v_0)(u_1, v_1)(u_2, v_2)\cdots$ of $\Gamma_f(L)$ for some f to be consistent with τ, if u_i is the prefix of $\tau(u_0 \cdots u_{i-1}, v_0 \cdots v_{i-1})$ of length $f(i)$. An i.o.t. strategy has access to both players' moves thus far, but cannot reconstruct when the moves of Player O were made.
4. A *history-tracking* (h.t.) strategy is a mapping $\tau\colon \Sigma_O^* \times (\mathbb{N}_+)^* \to \Sigma_I^\omega$. Again, consider a play $(u_0, v_0)(u_1, v_1)(u_2, v_2)\cdots$ of $\Gamma_f(L)$ for some f: it is consistent with τ, if u_i is the prefix of length $f(i)$ of $\tau(v_0 \cdots v_{i-1}, f(0)\cdots f(i-1))$. A h.t. strategy has access to the opponent's moves and to the values of the delay function for all previous rounds, which allows him to reconstruct his moves. Thus, giving him additionally access to his previous moves does not increase the strength of such a strategy.

As usual, we say that a strategy (of any type) is winning for Player I in $\Gamma_f(L)$ if the outcome of every play that is consistent with the strategy is in the complement of L. A strategy is omnipotent for L, if it is winning for Player I in $\Gamma_f(L)$ for every f.

The definitions above are given in order of increasing expressiveness, e.g., every (omnipotent) o.t. strategy can be turned into an (omnipotent) l.c. strategy inducing the same plays. The first two constructions are straightforward, and for the last one, Player I has to reconstruct his moves $u_0, \ldots, u_{i-1}$ using the information about the values $f(0), \ldots, f(i-1)$ and knowledge of his own i.o.t. strategy.

Our first result shows that the first three types of strategies form a strict hierarchy in terms of the games that can be won with them. The last case will be discussed in Section 7: it is open whether omnipotent h.t. strategies are strictly stronger than omnipotent i.o.t. strategies.

▶ **Theorem 6.** *There are winning conditions L_1 and L_2 such that*
1. *Player I has an omnipotent l.c. strategy for L_1, but no omnipotent o.t. strategy, and*
2. *Player I has an omnipotent i.o.t. strategy for L_2, but no omnipotent l.c. strategy.*

Proof. 1.) Let $L_1 = \{\binom{\alpha}{\beta} \mid \alpha \neq (ab)^\omega\}$. Intuitively, Player I wins a game with winning condition L_1 if he is able to produce the word $(ab)^\omega$, the moves of Player O are irrelevant. Indeed, one can easily build a l.c. strategy τ by defining $\tau(x, n) = (ab)^\omega$ for even n and $\tau(x, n) = (ba)^\omega$ for odd n. Every outcome of a play that is consistent with τ has $(ab)^\omega$ in its first component and is therefore winning for Player I. Hence, τ is omnipotent for L_1.

However, we claim that Player I has no omnipotent o.t. strategy τ for L_1. If $\tau(\varepsilon) \neq (ab)^\omega$, then τ is losing for some f such that $f(0)$ is larger than the first position where $\tau(\varepsilon)$ and $(ab)^\omega$ differ. Thus, we can assume $\tau(\varepsilon) = (ab)^\omega$. Now, fix some letter $c \in \Sigma_O$ and consider the first letter of $\tau(c)$: if it is a (the other case is dual), then τ is losing for every f with odd $f(0)$, as the first component of the resulting outcome contains two a's in a row, if Player O picks c in the first round.

2.) Fix $\Sigma_I = \{a, b, c\}$ and $\Sigma_O = \{b, c\}$, and define

$$L_2 = (\Sigma_I \times \Sigma_O)^\omega \setminus \left\{ \binom{\alpha}{\beta} \mid \alpha \in a^{n_0}\, \beta(0)\, a^{n_1}\, \beta(1) \cdot \Sigma_I^\omega \text{ with } n_1 > n_0 \right\},$$

i.e., in order to win, Player I has to copy the first two letters picked by Player O and ensure to produce more a's between these two positions than before the first one. It is straightforward to show that the following i.o.t. strategy for Player I is omnipotent for L_2:

$$\tau(x, y) = \begin{cases} a^\omega & \text{if } x = \varepsilon, \\ x(0)\, a^\omega & \text{if } |x| = 1, \\ a^{n-k+1}\, x(1)\, a^\omega & \text{if } |x| > 1 \text{ and } y = a^n x(0) a^k, \\ a^\omega & \text{otherwise.} \end{cases}$$

Intuitively, Player I picks a's until Player O has picked her first letter, which is immediately copied by Player I. Then, he picks a's until he has access to the second letter picked by Player O and then continues picking a's until the second a-block is longer than the first one. Then, he copies Player O's second letter and only picks a's afterwards. Note that it is crucial for Player I to have access to his own previous moves to guarantee that the second a-block is longer than the first one and that he does not necessarily pick $x(1)$ in the second round.

Next, we show that Player I has no omnipotent l.c. strategy for L_2. Towards a contradiction, assume there is one, call it τ. We claim that $\tau(x, n)$ begins with aa, $ax(0)$, or $x(0)a$ for every input (x, n) with $2|x| \leq n$. This is straightforward for $x = \varepsilon$ and $n = 0$ (the cases (ε, n) with $n > 0$ are irrelevant), since $\tau(\varepsilon, 0)$ has to be equal to a^ω. If not, Player O would have a counterstrategy against τ w.r.t. some large enough f by picking c in round 0, if the first non-a letter in $\tau(\varepsilon, 0)$ is b and vice versa.

It remains to consider an input (x, n) satisfying $0 < 2|x| \leq n$. Consider a delay function f satisfying $f(0) = n - |x| + 1$, $f(i) = 1$ for i in the range $1 \leq i \leq n$, and $f(n + 1) = 2$. Now, use τ in $\Gamma_f(L_2)$ against Player O picking the letters of x in the first $|x|$ rounds: Player I picks $a^{f(0)}$ in the first round, and $\alpha(1), \ldots, \alpha(|x| - 1)$ during the next $|x| - 1$ rounds, while Player O picks $x(0), \ldots, x(|x| - 1)$. Note that we have $|a^{f(0)}\alpha(1) \cdots \alpha(|x| - 1)| = n$; the next letters picked by Player I are therefore the first two of $\tau(x, n)$.

We consider two cases: if $\alpha(j) = a$ for every j, then the next two letters picked by τ may contain only a's, or one a and the first letter of x, but not any other combination. Especially, the second letter of x may not yet be picked, since the resulting a-block would

be of length zero, which would results in a losing play. On the other hand, if there is a j such that $\alpha(j) = x(0)$ then, all other $\alpha(j')$ have to be equal to a, since the second a-block would again be too short otherwise. Thus, the first a-block has at least length $n - |x| + 1$, which implies that the second a-block has at most length $|x| - 2$ after round n. As we have $n - |x| + 1 \geq |x| + 2$, we conclude that the next two letters picked by τ have to be both an a.

To conclude, apply τ in $\Gamma_{f'}(L_2)$ for the delay function f' with $f'(i) = 2$ for every i against Player O picking b and c in the first two rounds. As shown above, τ will pick aa, ab, or ba in every round. Thus, the resulting outcome is losing for Player I, as he never picks a c. ◀

Note that the winning condition L_1 is ω-regular and even recognizable by a deterministic ω-automaton with reachability acceptance condition, and therefore in $\mathbf{\Sigma}_1$. Furthermore, the winning condition L_2 is not ω-regular, but recognizable by a deterministic ω-pushdown automaton with safety acceptance, and in $\mathbf{\Pi}_1$.

4.2 Omnipotent Strategies for Player O

Now, we consider universal strategies for Player O. The standard definition given in Section 6 is syntactically independent of a fixed delay function. However, the reconstruction of Player O's moves made in previous rounds depends on knowledge about f. This can be exploited to show that strategies for Player O that have access to the number of rounds played already are more powerful than strategies which do not. Formally, we consider two types of omnipotent strategies for Player O corresponding to the first two notions for Player I. The other two notions introduced for Player I are not necessary for Player O.

1. An *input-tracking* (i.t.) strategy is a map $\sigma\colon \Sigma_I^* \to \Sigma_O$. Let $(u_0, v_0)(u_1, v_1)(u_2, v_2)\cdots$ be a play of $\Gamma_f(L)$ for some f: it is consistent with σ, if $v_i = \sigma(u_0 \cdots u_i)$. Such a strategy cannot reconstruct Player O's previous moves and cannot even determine how many rounds were played already.
2. A *round-counting* (r.c.) strategy is a mapping $\sigma\colon \Sigma_I^* \times \mathbb{N} \to \Sigma_O$. This, time, we say that a play $(u_0, v_0)(u_1, v_1)(u_2, v_2)\cdots$ of $\Gamma_f(L)$ for some f is consistent with the strategy σ, if $v_i = \sigma(u_0 \cdots u_i, i)$. A r.c. strategy has access to the opponent's moves and the number of rounds played thus far.

Note the asymmetry between the counting strategies for Player I and Player O: Player I counts the number of letters he has picked thus far and therefore, as he has direct access to Player O's moves, the size of the lookahead. Player O counts the number of rounds, i.e., the number of letters she has picked thus far. Again, this allows her to determine the size of the lookahead, as she has access to Player I's moves. Omnipotency for Player O's strategies is defined as before. Also, as for Player I, every i.t. strategy can be turned into an r.c. strategy. Finally, r.c. strategies are more powerful than omnipotent i.t. ones.

▶ **Theorem 7.** *There is a winning condition L_3 such that Player O has an omnipotent r.c. strategy for L_3, but no omnipotent i.t. strategy.*

The proof is a variation of the analogue for Player I: the winning condition requires Player O to produce $(ab)^\omega$, which she can do, if she has access to the number of rounds already played, but she cannot do it without this information. Again, the distinguishing winning condition is very simple: it is ω-regular and even recognizable by a deterministic ω-automaton with safety acceptance condition, and therefore in $\mathbf{\Pi}_1$.

5 Borel Determinacy of Delay Games with Omnipotent Strategies

Now, we turn our attention to delay games without fixed delay functions and show that there is either a winning strategy for Player O for some f, or Player I wins for every f with the *same* omnipotent strategy. Then, we show the dual result for Player O.

We still use the notation introduced at the beginning of Section 3 and start by defining $\mathrm{skip}(L) = \bigcup_f \mathrm{shift}_f(L)$, where there union ranges over all delay functions f. Note that Player O loses a play in a game with winning condition $\mathrm{skip}(L)$ if she picks $\triangleright$ all but finitely often. Also, we have $\mathrm{skip}(L) = \{\binom{\alpha}{\beta} \in (\Sigma_I \times \Sigma_O^{\triangleright})^\omega \mid \binom{\alpha}{h(\beta)} \in L\}$.

The tight connection between delay games $\Gamma_f(L)$ for arbitrary f and the delay-free game $\Gamma(\mathrm{skip}(L))$ appears implicitly in the work by Holtmann et al. [7] and is made explicit below. We exploit these connections to prove determinacy of delay games with Borel winning conditions.

▶ **Theorem 8.** *Let L be Borel. Either, Player O wins $\Gamma_f(L)$ for some f or Player I has an omnipotent h.t. strategy for L.*

Proof. First, we show that $\mathrm{skip}(L)$ is Borel and then apply the connection between the games $\Gamma_f(L)$ for arbitrary f and $\Gamma(\mathrm{skip}(L))$.

Proving $\mathrm{skip}(L)$ to be Borel is analogous to the proof for $\mathrm{shift}_f(L)$, we just replace the intersections with U_f by intersections with $U = \mathrm{skip}((\Sigma_I \times \Sigma_O)^\omega)$ (which is also in $\mathbf{\Pi}_2$) and the definition of K' in the induction start is changed to

$$K' = \bigcup_{\binom{\alpha(0)}{\beta(0)}\cdots\binom{\alpha(k)}{\beta(k)} \in K} \left\{ \binom{x}{y} \mid y \in \triangleright^*\beta(0)\cdots\triangleright^*\beta(k) \text{ and } x \in \alpha(0)\cdots\alpha(k)\cdot\Sigma_I^{|y|-(k+1)} \right\}.$$

Thus, $\Gamma(\mathrm{skip}(L))$ is determined.

Next, we show that Player O wins $\Gamma_f(L)$ for some f, if she wins $\Gamma(\mathrm{skip}(L))$. Let σ' be a winning strategy for Player O in $\Gamma(\mathrm{skip}(L))$. We construct a delay function f and a winning strategy σ for Player O in $\Gamma_f(L)$ by simulating a play in $\Gamma_f(L)$ by a play in $\Gamma(\mathrm{skip}(L))$.

We begin by defining f. For $i \in \mathbb{N}$ let ℓ_i be the maximal number such that Player O picks at most i non-skip symbols during the first ℓ_i rounds in every play of $\Gamma(\mathrm{skip}(L))$ that is consistent with σ'. We claim that every ℓ_i is well-defined. Assume ℓ_i for some fixed i is not. Then, the play prefixes under consideration for defining ℓ_i form an infinite, but finitely branching tree. Hence, König's Lemma implies the existence of an infinite play that is consistent with σ' during which Player O all but finitely often picks $\triangleright$. This play is losing for her, thus contradicting σ' being a winning strategy.

By construction, if Player I has picked $\ell_i + 1$ letters in $\Gamma(\mathrm{skip}(L))$, then σ' has determined at least $i+1$ non-skip letters. Now, let $f(0) = \ell_0 + 1$ and $f(i+1) = (\ell_{i+1} + 1) - \sum_{j=0}^{i} f(j)$. It remains to define σ: assume Player I has picked $u_0, \ldots, u_i$ in rounds $i = 0, 1, \ldots, i$ with $|u_j| = f(j)$. Consider the play prefix in $\Gamma(\mathrm{skip}(L))$ during which Player I picks $u_0 \cdots u_i$ and Player O plays according to σ'. We define $\sigma(u_0 \cdots u_i)$ to be the i-th non-skip letter (starting with the 0-th letter) picked by Player O on this play prefix. This is well-defined by the definition of f.

Let $\binom{\alpha}{\beta}$ an outcome that is consistent with σ. A straightforward induction shows that there is a play in $\Gamma(\mathrm{skip}(L))$ that is consistent with σ' and has an outcome $\binom{\alpha}{\beta'}$ such that $\beta = h(\beta')$. Hence, σ' being a winning strategy implies $\binom{\alpha}{\beta'} \in \mathrm{skip}(L)$ and therefore $\binom{\alpha}{\beta} \in L$. Thus, σ is a winning strategy for Player O in $\Gamma_f(L)$.

To conclude, we show that Player I has an omnipotent h.t. strategy τ for L, if he wins $\Gamma(\mathrm{skip}(L))$. To this end, let $\tau' \colon (\Sigma_O^{\triangleright})^* \to \Sigma_I$ be a winning strategy for Player I in $\Gamma(\mathrm{skip}(L))$.

We define an h.t. strategy $\tau\colon \Sigma_O^* \times (\mathbb{N}_+)^* \to \Sigma_O^\omega$. Let $x \in \Sigma_O^*$ and $n_0 \cdots n_{i-1} \in (\mathbb{N}_+)^*$. Note that τ will only be applied to inputs $(x, n_0 \cdots, n_{i-1})$ where $|x| = i$. Thus, we restrict our attention to those inputs. Let

$$x' = \triangleright^{n_0-1} x(0) \triangleright^{n_1-1} x(1) \cdots \triangleright^{n_{i-1}-1} x(i-1) \in (\Sigma_O^\triangleright)^*$$

and define

$$\tau(x, n_0 \cdots n_{i-1}) = \tau'(x')\, \tau'(x'\triangleright)\, \tau'(x'\triangleright\triangleright)\, \tau'(x'\triangleright\triangleright\triangleright) \cdots,$$

i.e., the answers according to τ' to Player O picking $\triangleright$ ad infinitum after picking x'.

A straightforward induction shows that for every outcome $\binom{\alpha}{\beta}$ that is consistent with τ in $\Gamma_f(L)$ for some f, there is an outcome $\binom{\alpha}{\beta'}$ that is consistent with τ' such that $h(\beta') = \beta$. As τ' is winning for Player I in $\Gamma(\mathrm{skip}(L))$ we have $\binom{\alpha}{\beta'} \notin \mathrm{skip}(L)$ and thus $\binom{\alpha}{\beta} \notin L$. Hence, τ is winning for $\Gamma_f(L)$ for every f and therefore omnipotent for L. ◀

The second part of the proof above (the equivalence of the delay games and the delay-free game) works for arbitrary winning conditions. Hence, we obtain the following corollary.

▶ **Corollary 9.** *If $\Gamma(\mathrm{skip}(L))$ is determined, then either Player O wins $\Gamma_f(L)$ for some f or Player I has an omnipotent h.t. strategy for L.*

It is open whether these results hold for i.o.t. strategies as well. This is related to the strictness of the strategy hierarchy mentioned earlier: is there a winning condition L such that Player I has an omnipotent h.t. strategy for L, but no omnipotent i.o.t. strategy? We discuss this question in Section 7.

To conclude this section, let us consider the case where Player O wins $\Gamma_f(L)$ for every delay function f. Here, we apply a monotonicity argument: the larger the lookahead is, the more information Player O has at her disposal, which makes winning easier for her. Thus, if she wins w.r.t. every delay function, then she wins in particular without lookahead. A winning strategy for the delay-free game can be turned into a winning strategy w.r.t. every *larger* delay function. Thus, the omnipotent strategy for Player O mimics the behavior of a winning strategy for the delay-free game and ignores the additional information given by the lookahead.

Formally, we order delay functions by the amount of lookahead available for Player O at every round: we define $f \sqsubseteq f'$, if and only if $\sum_{j=0}^{i} f(j) \leq \sum_{j=0}^{i} f'(j)$ for every $i \in \mathbb{N}$, i.e., in every round, the lookahead granted by f' is at least as large as the one granted by f. A winning strategy for Player O w.r.t. f can easily be turned into one for f' by ignoring the additional information. Thus, we obtain the following monotonicity property.

▶ Remark. If $f \sqsubseteq f'$ and Player O wins $\Gamma_f(L)$, then also $\Gamma_{f'}(L)$.

Note that *winning* refers to winning strategies that may depend on f respectively f'. Nevertheless, we can use monotonicity to obtain omnipotent strategies by considering the $\sqsubseteq$-minimal delay function. More formally, if Player O wins $\Gamma_f(L)$ for every f, then she wins in particular the delay-free game $\Gamma(L)$, i.e., the game w.r.t. the $\sqsubseteq$-minimal delay-function $i \mapsto 1$. It is easy to see that a winning strategy σ' for Player O in $\Gamma(L)$ can be turned into an omnipotent r.c. strategy σ for L: defining $\sigma(x, i) = \sigma'(x(0) \cdots x(i-1))$ simulates the strategy σ' by ignoring the additional information gained due to the lookahead.

▶ **Theorem 10.** *Either, Player I wins $\Gamma_f(L)$ for some f or Player O has an omnipotent r.c. strategy for L.*

As shown in Theorem 7, such a strategy has to have access to the number of rounds already played, the theorem does not hold for i.t. strategies. Note however, that this results holds for arbitrary winning conditions L.

A similar construction works if Player O does not have an omnipotent strategy for L, but wins $\Gamma_f(L)$ for some f. Then, she can simulate a winning strategy for $\Gamma_f(L)$ in $\Gamma_{f'}(L)$ for every $f' \sqsupseteq f$.

6 Decidability

In this section, we consider decision problems regarding omnipotent strategies, i.e., we are interested in determining whether a given player has an omnipotent strategy for a given winning condition L.

We begin with ω-regular conditions represented by deterministic parity automata.

▶ **Theorem 11.** *The following problems are* ExpTime*-complete respectively in* NP ∩ co-NP*:*

1. *Given a deterministic parity automaton $\mathcal{A}$, does Player I have an omnipotent h.t. strategy for $L(\mathcal{A})$?*
2. *Given a deterministic parity automaton $\mathcal{A}$, does Player O have an omnipotent r.c. strategy for $L(\mathcal{A})$?*

Proof. 1.) Due to Theorem 8, Player I has an omnipotent h.t. strategy for $L(\mathcal{A})$ if and only if there is no f such that Player O wins $\Gamma_f(L(\mathcal{A}))$. Determining whether there is an f such that Player O wins $\Gamma_f(L(\mathcal{A}))$ is ExpTime-complete [10]. Hence, determinacy of ω-regular delay games w.r.t. fixed delay functions and closure of ExpTime under complements yields the desired result.

2.) Due to Theorem 10, Player O has an omnipotent r.c. strategy for $L(\mathcal{A})$ if and only if she wins $\Gamma(L(\mathcal{A}))$. This game can be encoded as a parity game in an arena of size $2|\mathcal{A}|$ that has the same colors as $\mathcal{A}$. The winner of this game is solvable in NP ∩ co-NP (and even UP ∩ co-UP [9]), which yields the desired result. ◀

An omnipotent r.c. strategy for Player O can be implemented by a finite automaton with output of size $\mathcal{O}(|\mathcal{A}|)$, e.g., if the input $(x, n) \in \Sigma_I^* \times \mathbb{N}$ with $|x| \geq n$ is encoded as $x(0) \cdots x(n-1) \# x(n) \cdots x(|x|-1)$, where $\#$ is a fresh symbol. The states of the automaton are the vertices of the parity game constructed in the proof above and the output function is given by a positional winning strategy for this game.

Now, we turn our attention to ω-context-free winning conditions. Such languages are recognized by ω-pushdown automata, classical pushdown-automata running on infinite words. We refer to [2] for detailed definitions. First, we consider deterministic automata.

▶ **Theorem 12.** *The following problems are undecidable respectively* ExpTime*-complete:*

1. *Given a deterministic ω-pushdown automaton $\mathcal{A}$, does Player I have an omnipotent h.t. strategy for $L(\mathcal{A})$?*
2. *Given a deterministic ω-pushdown automaton $\mathcal{A}$, does Player O have an omnipotent r.c. strategy for $L(\mathcal{A})$?*

Proof. Recall that delay games with winning conditions that are recognized by deterministic ω-pushdown automata and w.r.t. fixed delay functions are determined [6].

1.) As in the ω-regular case, Player I has an omnipotent h.t. strategy for $L(\mathcal{A})$ if and only if there is no f such that Player O wins $\Gamma_f(L(\mathcal{A}))$. Determining whether there is an f such that Player O wins $\Gamma_f(L(\mathcal{A}))$ is undecidable [6]. Hence, determinacy w.r.t. fixed delay functions implies undecidability of the problem.

2.) Again, as in the ω-regular case, the problem can be reduced to solving the delay-free game $\Gamma(L(\mathcal{A}))$, which is ExpTime-complete [17]. ◀

As before, an omnipotent r.c. strategy for Player O can be represented finitely by constructing a pushdown-automaton with output that implements a winning strategy for the delay-free game $\Gamma(L(\mathcal{A}))$, which can be constructed effectively [17].

To conclude, we consider non-deterministic ω-pushdown automata.

▶ **Theorem 13.** *The following problems are undecidable:*

1. *Given a non-deterministic ω-pushdown automaton $\mathcal{A}$, does Player I have an omnipotent l.c. (i.o.t., h.t.) strategy for $L(\mathcal{A})$?*
2. *Given a non-deterministic ω-pushdown automaton $\mathcal{A}$, does Player O have an omnipotent r.c. strategy for $L(\mathcal{A})$?*

Proof. Recall that the (non-)universality problem for non-deterministic ω-pushdown automata is undecidable (see, e.g., [4]). Given such an automaton $\mathcal{A}$, we define the winning condition $I_{\mathcal{A}} = \{\binom{\alpha}{\alpha} \mid \alpha \in L(\mathcal{A})\}$, i.e., in order to win, Player I has to produce an $\alpha \notin L(\mathcal{A})$.

1). We prove undecidability for the case of l.c. strategies by a reduction from the non-universality problem. The other cases are proven similarly.

We claim that $L(\mathcal{A})$ is non-universal if and only if Player I has an omnipotent l.c. strategy for $I_{\mathcal{A}}$. Let $L(\mathcal{A})$ be non-universal, i.e., we can fix some $\alpha \notin L(\mathcal{A})$. Then, there is a l.c. strategy for Player I that produces α, independently of the moves of Player O. Hence, this strategy is omnipotent for $I_{\mathcal{A}}$. Now, if $I_{\mathcal{A}}$ is universal then Player O wins $\Gamma(I_{\mathcal{A}})$ by just copying Player I's moves. Hence, Player I has no omnipotent strategy for $I_{\mathcal{A}}$.

2.) We claim that $L(\mathcal{A})$ is universal if and only if Player O has an omnipotent r.c. strategy for $I_{\mathcal{A}}$. Above, we have shown that $I_{\mathcal{A}}$ being universal implies that Player O wins $\Gamma_f(I_{\mathcal{A}})$ for every f. Hence, due to Theorem 10, Player O has an omnipotent r.c. strategy for $I_{\mathcal{A}}$. On the other hand, as seen above, if $I_{\mathcal{A}}$ is non-universal, then Player I has an omnipotent strategy for $I_{\mathcal{A}}$, which implies that $I_{\mathcal{A}}$ has none. ◀

It is open whether the problems asking for weaker types of omnipotent strategies are decidable. We discuss these problems in the next section.

7 Characterizing the Existence of Omnipotent Strategies

In this section, we give a characterization of omnipotent strategies for delay games in terms of uniform strategies for delay-free games. We focus on the case of i.o.t. strategies for Player I, but the other cases are analogous.

Fix some strategy $\tau\colon (\Sigma_O^{\triangleright})^* \to \Sigma_I$ and define the equivalence relation $\approx_\tau$ over $(\Sigma_O^{\triangleright})^*$ via $x_0 \approx x_1$ if and only if $|x_0| = |x_1|$, $h(x_0) = h(x_1)$, and $\tau(x_0') = \tau(x_1')$ for all proper prefixes $x_0' \sqsubset x_0$ and $x_1' \sqsubset x_1$ with $|x_0'| = |x_1'|$. Thus, x_0 and x_1 are equivalent, if Player O has picked the same sequence of non-skip symbols in x_0 and in x_1, has picked the same number of skip symbols (but possibly at different positions), and τ picked the same moves answering to Player O picking x_0 and x_1, respectively, during the previous rounds.

Now, we say that a strategy τ for Player I in $\Gamma_f(\mathrm{skip}(L))$ is i.o.-uniform if $\tau(x) = \tau(x')$ for all $x \approx_\tau x'$. The following lemma is a straightforward extension of Theorem 8.

▶ **Lemma 14.** *Player I has an omnipotent i.o.t. strategy if and only if Player I has an i.o.-uniform winning strategy for $\Gamma(\mathrm{skip}(L))$.*

We conjecture that Player I always has such a uniform strategy.

▶ **Conjecture 15.** *If Player I wins $\Gamma(\mathrm{skip}(L))$, then she has an i.o.-uniform winning strategy for $\Gamma(\mathrm{skip}(L))$.*

Note that we do not impose any requirements on L. If the conjecture is true, then Theorem 8 is also true for i.o.t. strategies.

The existence of an omnipotent o.t., l.c., or i.t. strategy for a winning condition L can be characterized analogously using appropriate equivalence relations that capture the limited access to information about the history of a play that such a strategy has.

Furthermore, the existence of such uniform strategies can be expressed in the framework introduced by Maubert and Pinchinat [12]: they investigate infinite games under uniformity constraints on strategies expressed in an extension of LTL with a modality to equate finite play prefixes that are in some given equivalence relation. The logic is able to express the uniformity constraint formulated above, but our problems are not in the decidable fragment presented in this work, as the equivalence relations that characterize universal strategies are not rational (recognizable by an asynchronous transducer) and turning an ω-regular L into $\mathrm{skip}(L)$ does not preserve ω-regularity.

8 Conclusion

We presented determinacy results for delay games with Borel winning conditions, both with and without respect to fixed delay functions: in the latter case, we showed the existence of omnipotent strategies, i.e., strategies that are winning w.r.t. every delay function. In particular, we analyzed the exact amount of information such a strategy needs about the history of the play and the delay function under consideration. For games w.r.t. a fixed delay function, on which winning strategies may depend, access to the opponent's moves is sufficient. However, for omnipotent strategies the situation is more intricate: Player O needs access to the opponent's moves and the number of rounds played thus far, just having access to the opponent's moves is not sufficient. For Player I, we showed that access to both player's moves is necessary and having the full information about the play's history is trivially sufficient. However, it is open whether that much information is necessary: does access to both player's previous moves, but not to the delay function under consideration, suffice to implement an omnipotent strategy? To answer this question, we currently work on resolving Conjecture 15.

Also, we determined the precise computational complexity of decision problems of the following form for ω-regular and ω-context-free winning conditions: given a winning condition L, does Player $p \in \{I, O\}$ have an omnipotent strategy for L?

Another interesting question concerns the decision problems left open in Section 6: can one decide if Player I has an omnipotent o.t. (l.c., i.o.t.) strategy for a given ω-regular winning condition? The analogous question for Player O and input-tracking strategies is also open. Furthermore, we left open the finite representability of omnipotent strategies for Player I for ω-regular winning conditions. We expect the techniques we developed to give an exponential-time algorithm for solving ω-regular delay games [10] to yield such strategies, but this is beyond the scope of this paper.

Another interesting open problem is to develop a theory of finite-state and positional winning strategies for delay games, both for the case with a fixed delay function and the universal case, and to prove positional respectively finite-state determinacy results.

Acknowledgments. The work presented here was initiated by a discussion with Dietmar Berwanger at the Dagstuhl Seminar "Non-Zero-Sum-Games and Control" in 2015.

References

1 Alonzo Church. Applications of recursive arithmetic to the problem of circuit synthesis. In *Summaries of the Summer Institute of Symbolic Logic*, volume 1, pages 3–50. Cornell University, 1957.

2 Rina S. Cohen and Arie Y. Gold. ω-computations on deterministic pushdown machines. *J. Comput. Syst. Sci.*, 16(3):275–300, 1978.

3 E. Allen Emerson and Charanjit S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *FOCS 1991*, pages 368–377. IEEE, 1991.

4 Olivier Finkel. Topological properties of omega context-free languages. *Theor. Comput. Sci.*, 262(1):669–697, 2001.

5 Olivier Finkel. The determinacy of context-free games. *J. Symb. Log.*, 78(4):1115–1134, 2013.

6 Wladimir Fridman, Christof Löding, and Martin Zimmermann. Degrees of lookahead in context-free infinite gmes. In Marc Bezem, editor, *CSL 2011*, volume 12 of *LIPIcs*, pages 264–276. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2011.

7 Michael Holtmann, Łukasz Kaiser, and Wolfgang Thomas. Degrees of lookahead in regular infinite games. *LMCS*, 8(3), 2012.

8 Frederick A. Hosch and Lawrence H. Landweber. Finite delay solutions for sequential conditions. In *ICALP 1972*, pages 45–60, 1972.

9 Marcin Jurdziński. Deciding the winner in parity games is in UP $\cap$ CO-UP. *Inf. Process. Lett.*, 68(3):119–124, 1998.

10 Felix Klein and Martin Zimmermann. How much lookahead is needed to win infinite games? In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *ICALP 2015*, volume 9135 of *LNCS*, pages 452–463. Springer, 2015.

11 Donald A. Martin. Borel determinacy. *Annals of Mathematics*, 102:363–371, 1975.

12 Bastien Maubert and Sophie Pinchinat. A general notion of uniform strategies. *IGTR*, 16(1), 2014.

13 Robert McNaughton. Playing infinite games in finite time. In Arto Salomaa, Derick Wood, and Sheng Yu, editors, *A Half-Century of Automata Theory*, pages 73–91. World Scientific, 2000.

14 Andrzej Mostowski. Games with forbidden positions. Technical Report 78, University of Gdańsk, 1991.

15 Wolfgang Thomas and Helmut Lescow. Logical specifications of infinite computations. In J. W. de Bakker, Willem P. de Roever, and Grzegorz Rozenberg, editors, *REX 1993*, volume 803 of *LNCS*, pages 583–621. Springer, 1993.

16 B.A. Trakhtenbrot and I.M. Barzdin. *Finite Automata; Behavior and Synthesis*. Fundamental Studies in Computer Science, V. 1. North-Holland Publishing Company; New York: American Elsevier, 1973.

17 Igor Walukiewicz. Pushdown processes: Games and model-checking. *Inf. Comput.*, 164(2):234–263, 2001.

18 Ernst Zermelo. über eine Anwendung der Mengenlehre auf die Theorie des Schachspiels. In *Proceedings of the Fifth Congress of Mathematicians, Vol. 2*, pages 501–504. Cambridge Press, 1913.

19 Martin Zimmermann. Delay games with WMSO+U winning conditions. In Lev D. Beklemishev and Daniil V. Musatov, editors, *CSR 2015*, volume 9139 of *LNCS*, pages 412–425. Springer, 2015.

On Unambiguous Regular Tree Languages of Index (0,2)

Jacques Duparc[1], Kevin Fournier[1,2], and Szczepan Hummel*[3]

1 **Université de Lausanne, Switzerland**
jacques.duparc@unil.ch
2 **Université Paris Diderot, France**
kevin.fournier@imj-prg.fr
3 **University of Warsaw, Poland**
shummel@mimuw.edu.pl

Abstract

Unambiguous automata are usually seen as a natural class of automata in-between deterministic and nondeterministic ones. We show that in case of infinite tree languages, the unambiguous ones are topologically far more complicated than the deterministic ones. We do so by providing operations that generate a family $\left(\mathcal{A}^b_\alpha\right)_{\alpha<\varphi_2(0)}$ of unambiguous automata such that:

1. It respects the strict Wadge ordering: $\alpha < \beta$ if and only if $\mathcal{A}^b_\alpha <_W \mathcal{A}^b_\beta$. This can be established without the help of any determinacy principle, simply by providing effective winning strategies in the underlying games.
2. Its length ($\varphi_2(0)$) is the first fixpoint of the ordinal function that itself enumerates all fixpoints of the ordinal exponentiation $x \mapsto \omega^x$: an ordinal tremendously larger than $(\omega^\omega)^3 + 3$ which is the height of the Wadge hierarchy of *deterministic* tree languages as uncovered by Filip Murlak.
3. The priorities of all these parity automata only range from 0 to 2.

1998 ACM Subject Classification F.1.1 Models of Computation, F.1.3 Complexity Measures and Classes, F.4.1 Mathematical Logic

Keywords and phrases Tree Automata, Unambiguity, Wadge Hierarchy

Digital Object Identifier 10.4230/LIPIcs.CSL.2015.534

1 Introduction

An unambiguous automaton is a nondeterministic automaton that admits at most one accepting run on each input. By definition, the class of languages recognized by unambiguous automata includes the class of languages recognized by deterministic automata and is included in the class of languages recognized by nondeterministic automata. Depending on the context, some of these inclusions may be strict. For example, in the case of finite automata on finite words, none of these inclusions is strict, because every regular language is recognized by a deterministic finite automaton. The picture is still trivial for infinite words if we consider the parity condition, but becomes more interesting for Büchi automata. While not every ω-regular (nondeterministic) language is recognized by deterministic Büchi automaton, it always can be recognized by an unambiguous automaton ([2]). From the algorithmic perspective unambiguous automata may be considered as a trade off between succinctness

* The author was partially supported by Polish National Science Centre grant no. 2014-13/B/ST6/03595.

24th EACSL Annual Conference on Computer Science Logic (CSL 2015).
Editor: Stephan Kreutzer; pp. 534–548

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

and efficiency. It is best understood for finite words. One can find example of unambiguous automaton exponentially more succinct than the corresponding deterministic one, while universality, equivalence and inclusion decision problems are in P for unambiguous and are $PSPACE$-complete for nondeterministic automata (see [9] for more details and further references).

In this paper we concentrate on infinite trees and parity condition. On the one hand, it is easy to observe that unambiguous automata are more expressive than the deterministic ones in this context: consider for example the language "exists exactly one branch with infinitely many labels a". On the other hand, it took a while to determine whether there are regular languages that are inherently ambiguous: it was shown by Niwiński and Walukiewicz in [14] (later described in [7] and [8]) that unambiguous automata do not recognize all nondeterministic languages. Still, unambiguous automata, although poorly understood so far, can occur to be an important intermediate model, as many questions seem to be very hard to answer for nondeterministic automata. For example no algorithm is known that calculates Rabin-Mostowski index for a given regular language.

The tool to measure the position of unambiguous languages of infinite trees in between deterministic and nondeterministic ones is given by descriptive set theory through the notion of *topological complexity*. It is well known that deterministic parity tree automata recognize only languages in the $\mathbf{\Pi}^1_1$ class (coanalytic sets), whereas nondeterministic automata recognize some languages that are neither analytic, nor coanalytic. The expressive power of nondeterministic automata is nonetheless bounded by the second level of the projective hierarchy, and, by Rabin's complementation result ([15]), all nondeterministic languages are in fact in the $\mathbf{\Delta}^1_2$ class. In [12], the third author gives an unambiguous language G which is $\mathbf{\Sigma}^1_1$-complete, and constructs from it an unambiguous language that is outside both $\mathbf{\Pi}^1_1$ and $\mathbf{\Sigma}^1_1$. A finer topological complexity measure is therefore needed: the Wadge hierarchy, which relies on the notion of reductions by continuous functions (Wadge-reducibility). Complexity classes, called Wadge degrees, consist of sets Wadge-reducible to each other, and constitute a hierarchy whose levels, called ranks, can be enumerated with ordinals. We describe a series of operations on automata that preserve unambiguity and lift the Wadge degrees of the recognized languages. We emphasize that this is done without any particular determinacy principle. In particular, we do not require $\mathbf{\Delta}^1_2$-determinacy. These operations help us generate a hierarchy of canonical unambiguous languages of higher and higher topological complexity. This hierarchy has $\varphi_2(0)$ many levels, where $\varphi_2(0)$ stands for the first fixpoint of the ordinal function[1] $x \mapsto \varepsilon_x$ which itself enumerates the fixpoints of the exponentiation $x \mapsto \omega^x$. Compared to the height of the Wadge hierarchy of all deterministic tree languages, which is $(\omega^\omega)^3 + 3$ as established by Filip Murlak in [13], the ordinal $\varphi_2(0)$ is tremendously larger.

The gap between the respective topological complexity of the two considered classes of languages, measured by the difference between the height of their respective Wadge hierarchies, illustrates the discrepancy between these classes. It is nonetheless not the only interest of the descriptive set theoretic framework, as it is shown by the recent results obtained on MSO+U. In [5], a topological complexity result was used to prove that there is no algorithm that decides satisfiability of a formula of MSO+U logic on infinite trees and that has a correctness proof in ZFC (an "almost undecidability" result), partially answering a question that was open for over ten years [4]. The elementary undecidability argument was later given in [6], but the topological result came first and motivated the research towards

[1] not to be mistaken with an ε-move.

the other. Our constructions provide benchmarks for the study of unambiguous languages, and could lead to prominent algorithmic results for this class. It might, for example, help determine if it is decidable whether a given nondeterministic language is unambiguous. The result also could contribute to solving unambiguous index problem as it can help in characterising unambiguous languages of index $(0, 2)$.

2 Preliminaries

2.1 The Wadge hierarchy and the Wadge game

The Wadge theory is in essence the theory of *pointclasses* (see [1]). Let X be a topological space. A pointclass is a collection of subsets of X that is closed under continuous preimages. For Γ a pointclass, we denote by $\check{\Gamma}$ its *dual* class containing all the subsets of X whose complements are in Γ, and by $\Delta(\Gamma)$ the ambiguous class $\Gamma \cap \check{\Gamma}$. If $\Gamma = \check{\Gamma}$, we say that Γ is self-dual.

The *Wadge preorder* $\leq_W$ on $\mathscr{P}(X)$ is defined as follows: for $A, B \subseteq X$, $A \leq_W B$ if and only if there exists $f : X \longrightarrow X$ continuous such that $f^{-1}(B) = A$. It is merely by definition a preorder. The Wadge preorder induces an equivalence relation $\equiv_W$ whose equivalence classes are called the *Wadge degrees*, and denoted by $[A]_W$. We say that the set $A \subseteq X$ is *self-dual* if it is Wadge equivalent to its complement, that is if $A \equiv_W A^{\complement}$, and *non self-dual* if it is not. We use the same terminology for the Wadge degrees.

Let Γ be a pointclass of X. There is a strong connection between pointclasses included in Γ and Wadge degrees of sets in Γ, since all non self-dual pointclasses are of the form

$$\{B \subseteq X : B \leq_W A\}$$

for some non self-dual set A, while self-dual pointclasses are all of the form

$$\{B \subseteq X : B \leq_W A \text{ and } A \not\leq_W B\},$$

also for some non self-dual set A. We have thus a direct correspondence between $(\mathscr{P}(X), \leq_W)$ restricted to Γ and the pointclasses included in Γ with the inclusion: the pointclasses are exactly the initial segments of the Wadge preorder. In particular, the Wadge hierarchy refines tremendously the Borel and the Projective hierarchies.

A *conciliatory* binary tree over a finite set Σ is a partial function $t : \{0,1\}^* \to \Sigma$ with a prefix closed domain. Those trees can have both infinite and finite branches. A tree is called *full* if $\mathrm{dom}(t) = \{0,1\}^*$. Let $\mathcal{T}_\Sigma^{\leq\omega}$ and T_Σ denote, respectively, the set of all conciliatory binary trees and the set of full binary trees over Σ. Given $x \in \mathrm{dom}(t)$, we denote by t_x the subtree of t rooted in x. Let $\{0,1\}^n$ denote the set of words over $\{0,1\}$ of length n, and let t be a conciliatory tree over Σ. We denote by $t[n]$ the finite initial binary tree of height $n+1$ given by the restriction of t to $\bigcup_{0\leq i\leq n}\{0,1\}^i$.

The space T_Σ equipped with the standard Cantor topology is a Polish space and is in fact homeomorphic to the Cantor space[2]. Let $L, M \subseteq T_\Sigma$, the Wadge game $W(L, M)$ is a two player infinite game that provides a very useful characterization for the Wadge preorder. In this game, each player builds a tree, say t_{I} and t_{II}. At every round, player I plays first, and both players add a finite number of children to the terminal nodes of their tree. Player II is allowed to skip its turn, but has to produce a tree in T_Σ throughout a game. Player II wins the game if and only if $t_I \in L \Leftrightarrow t_{II} \in M$.

[2] See for example [3].

▶ **Lemma 1** ([18]). *Let $L, M \subseteq T_\Sigma$. Then $L \leq_W M$ if and only if player* II *has a winning strategy in the game $W(L, M)$.*

We write $A <_W B$ when II has a winning strategy in $W(A, B)$ *and* I has a winning strategy in $W(B, A)$[3].Given a pointclass Γ of T_Σ with suitable closure properties, the assumption of the determinacy of Γ is sufficient to prove that Γ is semi-linearly ordered by $\leq_W$, denoted $\mathrm{SLO}(\Gamma)$, i.e. that for all $L, M \in \Gamma$,

$$L \leq_W M \qquad \text{or} \qquad M \leq_W L^{\mathsf{C}},$$

and that $\leq_W$ is well founded when restricted to sets in Γ ([16, 1]). Under these conditions, the Wadge degrees of sets in Γ with the induced order is thus a hierarchy called the *Wadge hierarchy*. Therefore, there exists a unique ordinal, called the height of the Γ-Wadge hierarchy, and a mapping d_W^Γ from the Γ-Wadge hierarchy onto its height, called the *Wadge rank*, such that, for every L, M non-self-dual in Γ, $d_W^\Gamma(L) < d_W^\Gamma(M)$ if and only if $L <_W M$ and $d_W^\Gamma(L) = d_W^\Gamma(M)$ if and only if $L \equiv_W M$ or $L \equiv_W M^{\mathsf{C}}$. The wellfoundedness of the Γ-Wadge hierarchy ensures that the Wadge rank can be defined by induction as follows:

- $d_W^\Gamma(\varnothing) = d_W^\Gamma(\varnothing^{\mathsf{C}}) = 1$
- $d_W^\Gamma(L) = \sup\left\{d_W^\Gamma(M) + 1 : M \text{ is non-self-dual}, M <_W L\right\}$ for $L >_W \varnothing$.

Note that given two pointclasses Γ and Γ', for every $L \in \Gamma \cap \Gamma'$, $d_W^\Gamma(L) = d_W^{\Gamma'}(L)$. Under sufficient determinacy assumptions, we can therefore safely speak of *the* Wadge rank of a tree language, denoted by d_W, as its Wadge rank with respect to any topological class including it. However the main result of this article does not provide any Wadge rank for the canonical languages that are constructed, because we do not make use of any determinacy principle.

2.2 The Conciliatory Hierarchy

For conciliatory languages L, M we define the *conciliatory* version of the Wadge game: $C(L, M)$ ([10, 11]). The rules are similar, except for the fact that both players are now allowed to skip and to produce trees with finite branches - or even finite trees. For conciliatory languages L, M we use the notation $L \leq_c M$ if and only if II has a winning strategy in the game $C(L, M)$. If $L \leq_c M$ and $M \leq_c L$, we will write $L \equiv_c M$. The conciliatory hierarchy is thus the partial order induced by $\leq_c$ on the equivalence classes given by $\equiv_c$. We write $A <_c B$ when II has a winning strategy in $C(A, B)$ *and* I has a winning strategy in $C(B, A)$.

From a conciliatory language L over Σ, one defines the corresponding language L^b of full trees over $\Sigma \cup \{b\}$ by:

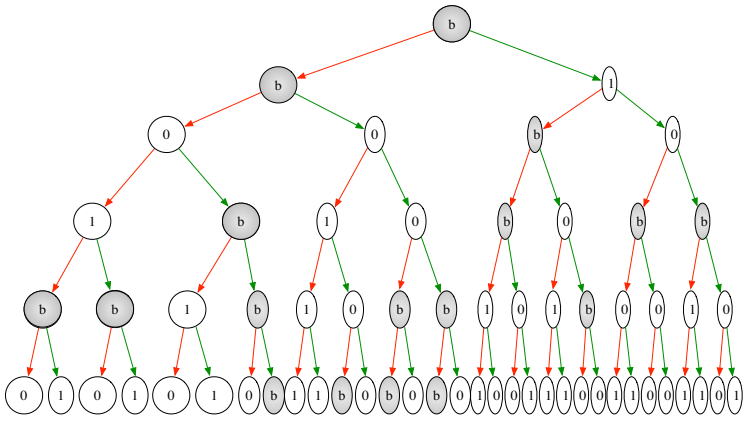
$$L^b = \left\{t \in T_{\Sigma\cup\{b\}} : t_{[\ /b]} \in L\right\},$$

where b is an extra symbol that stands for "blank", and $t_{[\ /b]}$, the *undressing* of t, is informally the conciliatory tree over Σ obtained once all the occurences of b have been removed in a top-down manner. More precisely, if there is a node v such that $t(v) = b$, we ignore this node and replace it with $v0$. If, for every integer n, $t(v0^n) = b$, then $v \notin \mathrm{dom}(t_{[\ /b]})$. This process is illustrated by Figure 1.

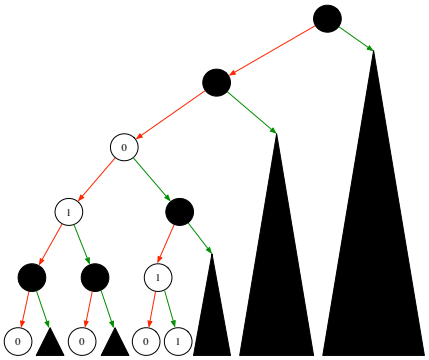
Formally, for each $v \in \{0,1\}^*$ we consider two (possibly infinite) sequences (w_i) and (u_i) in $\{0,1\}^*$:

- $w_0 = \varepsilon$, $u_0 = v$,

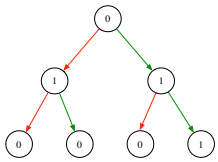
[3] This is in general stronger than the usual $A <_W B$ if and only if $A \leq_W B$ and $B \not\leq_W A$, but the two definitions coincide when the classes considered are determined.



(a) A tree t with blanks



(b) The blanks are deleted in a top-down manner



(c) The resulting tree $t_{[\ /b]}$

Figure 1 The undressing process.

- for $0 \leq i$:
  - if $t(w_i) = b$, we set $u_{i+1} = u_i$ and $w_{i+1} = w_i 0$;
  - if $t(w_i) \neq b$ and $u_i = a u_i'$ for $a \in \{0,1\}$, we set $u_{i+1} = u_i'$ and $w_{i+1} = w_i a$;
  - if $t(w_i) \neq b$ and $u_i = \varepsilon$, we halt the construction at step i.

If the construction is halted at some step i, then $v \in \mathrm{dom}(t_{[\ /b]})$ and $t_{[\ /b]}(v) = t(w_i)$. Otherwise, $v \notin \mathrm{dom}(t_{[\ /b]})$. If Γ is a pointclass of full trees, we say that a conciliatory language L is in Γ if and only if L^b is in Γ.

▶ **Lemma 2.** *Let L and M be conciliatory languages. Then*

$L \leq_c M$ *if and only if* $L^b \leq_W M^b$.

Proof. A strategy in one game can be translated directly into a strategy in the other game: arbitrary skipping in $C(L, M)$ gives the same power as the b labels in $W(L^b, M^b)$. In particular, in $W(L^b, M^b)$, II does not need to skip at all. ◀

The mapping $L \mapsto L^b$ gives thus a natural embedding of the preorder $\leq_c$ restricted to conciliatory sets in Γ into the Γ-Wadge hierarchy. Hence, for Γ with suitable closure and determinacy properties, the conciliatory degrees of sets in Γ with the induced order constitute a hierarchy called the *conciliatory hierarchy*. We define, by induction, the corresponding *conciliatory rank* of a language:

- $d_c^\Gamma(\varnothing) = d_c^\Gamma(\varnothing^{\complement}) = 1$
- $d_c^\Gamma(L) = \sup\{d_c^\Gamma(M) + 1 : M <_c L\}$ for $L >_c \varnothing$.

Similarly to the Wadge case, given two pointclasses Γ and Γ', for every conciliatory $L \in \Gamma \cap \Gamma'$, $d_c^\Gamma(L) = d_c^{\Gamma'}(L)$. Under sufficient determinacy assumptions, we can therefore safely speak, of *the* conciliatory rank of a conciliatory tree language, denoted by d_c, as its conciliatory rank with respect to any topological class including it. Observe that the conciliatory hierarchy does not contain self-dual languages: a strategy for I in $C(L, L^{\complement})$ is to skip in the first round, and then copy moves of II.

2.3 Automata and conciliatory trees

A *nondeterministic parity tree automaton* $\mathcal{A} = \langle \Sigma, Q, I, \delta, \mathrm{r} \rangle$ consists of a finite input alphabet Σ, a finite set Q of states, a set of initial states $I \subseteq Q$, a transition relation $\delta \subseteq Q \times \Sigma \times Q \times Q$ and a priority function $\mathrm{r} : Q \to \omega$. A run of automaton $\mathcal{A}$ on a binary conciliatory input tree $t \in \mathcal{T}_\Sigma^{\leq\omega}$ is a conciliatory tree $\rho_t \in \mathcal{T}_Q^{\leq\omega}$ with $\mathrm{dom}(\rho_t) = \{\varepsilon\} \cup \{va : v \in \mathrm{dom}(t) \wedge a \in \{0,1\}\}$ such that the root of this tree is labeled with a state $q \in I$, and for each $v \in \mathrm{dom}(t)$, transition $(\rho_t(v), t(v), \rho_t(v_1), \rho_t(v_1)) \in \delta$. The run ρ_t is *accepting* if parity condition is satisfied on each infinite branch of ρ_t, i.e. if the highest rank of a state occurring infinitely often on the branch is even, and if the rank of each leaf node in ρ_t is even. We say that a parity tree automaton A *accepts* a conciliatory tree t if it has an accepting run on t. The language *recognized* by A, denoted $L(A)$ is the set of trees accepted by A. The Rabin-Mostowski index of the automaton is a pair $(\min(r(Q)), \max(r(Q)))$. A language is of index (i,k) if it is recognized by some automaton of index (i,k). An automaton is *unambiguous* if it has at most one accepting run on each input. We denote by $L^\omega(A)$ the set of full trees recognized by A, i.e. $L^\omega(A) = L(A) \cap T_\Sigma$.

▶ **Corollary 3.** *The mapping $L \mapsto L^b$ embeds the conciliatory hierarchy for $\mathbf{\Delta}^1_2$-sets restricted to unambiguously recognizable languages into the $\mathbf{\Delta}^1_2$-Wadge hierarchy restricted to unambiguously recognizable languages.*

Proof. By Lemma 2 it is enough to prove that each unambiguous automaton A can be transformed into an unambiguous automaton A' such that $L^\omega(A') = L(A)^b$. Given any unambiguous automaton A, this is done by adding an all-accepting state $\top$ to the set of states Q_A, and the set $\{(q, b, q, \top) : q \in Q_A\}$ to the transition relation δ_A. The obtained automaton A' is unambiguous and such that $L^\omega(A') = L(A)^b$. ◀

In the diagrams of automata below, we use ε-transitions, i.e transitions that change state but do not progress run down a tree. This is, however, only a notation shortcut here — we do it to emphasize nondeterministic choice better. The transitions can be easily simulated by adding more transitions of the type as in the above definition to the state they lead from. We also use the following conventions in the diagrams. Nodes represent states of the automaton. Node labels correspond to state ranks. We additionally mark parity of ranks by node colors: nodes corresponding to states with even ranks are green, while nodes corresponding to states with odd ranks are red. A red edge shows the state that is assigned to the left successor node of a transition, a green edge goes to the right successor node. Edge label marks the label of a tree the transition goes through. In order to lighten the notation, transitions that are not depicted on a diagram lead to some definitely all-accepting state.

3 Operations on languages and their automatic counterparts

In this section, we present classical operations ([11]) on conciliatory tree languages that allow us to construct more and more complicated languages, and we prove that they preserve unambiguity, i.e. that if we apply them to unambiguously recognizable languages, the resulting language is equivalent[4] to an unambiguously recognizable one. Without loss of generality, we may choose the alphabet $\Sigma = \{0, 1\}$.

3.1 The sum

For $L, M \subseteq \mathcal{T}_\Sigma^{\leq\omega}$, we define the *sum* of L and M, in symbols $L \oplus M$, as the conciliatory tree language containing all of those trees $t \in \mathcal{T}_\Sigma^{\leq\omega}$ that one of the following conditions holds:

- $t(10^n) = 0$ for each integer n and $t_0 \in M$;
- the node 10^n is the first on the path 10^* labeled with 1 and either $t(10^n0) = 0$ and $t_{10^n00} \in L$, or $t(10^n0) = 1$ and $t_{10^n00} \in L^{\mathsf{C}}$

This operation behaves well regarding the conciliatory hierarchy.

▶ **Facts 4** ([10, 11]). *Let L, M, and M' be conciliatory tree languages over Σ. Then the following hold.*

1. $(L \oplus M)^{\mathsf{C}} \equiv_c L \oplus M^{\mathsf{C}}$.
2. $(L \oplus M) \oplus M' \equiv_c L \oplus (M \oplus M')$.
3. *The operation $\oplus$ preserves the conciliatory ordering: if $M' \leq_c M$, then*

$$L \oplus M' \leq_c L \oplus M.$$

Let $\mathcal{A}$ and $\mathcal{B}$ be two automata that recognize respectively the conciliatory languages M and L. Then the automaton $\mathcal{B} + \mathcal{A}$ depicted in Figure 2 recognizes the sum of L and M. In this picture, $\mathcal{C}$ is any automaton that recognizes a language equivalent to L^{C}, and the parities i and j are defined as follows:

- $i = 0$ if and only if the empty tree is accepted by $\mathcal{A}$;
- $j = 1$ if and only if $L(\mathcal{A})$ is equivalent to $L(\mathcal{A}) \to \ominus$.[5]

Note that the operation sum is *in itself* unambiguous, so that if $\mathcal{A}$ and $\mathcal{B}$ are unambiguous, and if there exists an unambiguous $\mathcal{C}$ equivalent to the complement of $\mathcal{B}$, their sum $\mathcal{B} + \mathcal{A}$ is also unambiguous. The core observation here is that only one of the initial ε-transitions can be taken in a root of a given tree, depending on whether there is a node labeled with 1 on branch 10^* in the tree or not. Moreover, if L and M are unambiguously recognizable conciliatory languages, and if the complement of M is equivalent to an unambiguously recognizable language $\check{M}$, the complement of $L \oplus M$ is equivalent to $L \oplus \check{M}$, which is unambiguously recognizable.

For $M \subseteq \mathcal{T}_\Sigma^{\leq\omega}$ and $n \in \omega$, we denote by $M \odot n$ the sum of M with itself n times:

$$M \odot n = \underbrace{M \oplus M \oplus \ldots \oplus M}_{n \text{ times}}.$$

If $\mathcal{A}$ recognizes M, we denote by $\mathcal{A} \bullet n$ the automaton that recognizes $M \odot n$.

---

[4] Relatively to $\equiv_c$.

[5] A player in charge of $L(\mathcal{A}) \to \ominus$ in a conciliatory game is like a player in charge of $L(\mathcal{A})$, but with the extra possibility at any moment of the play to reach a definitively rejecting position. We denote by $\ominus$ the automaton that rejects all trees.

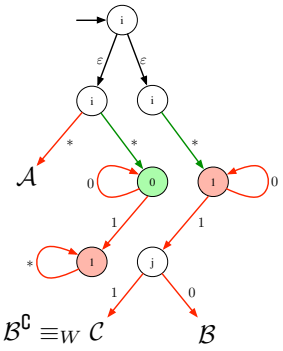


Figure 2 The automaton $\mathcal{B} + \mathcal{A}$ that recognizes $L(\mathcal{B}) \oplus L(\mathcal{A})$. The values of i and j depend on properties of $\mathcal{A}$. The transitions that are not depicted lead to an all-accepting state $\top$.

▶ **Lemma 5.** *Let L, L', M and M' be conciliatory languages such that $L <_c L'$ and $M \leq_c M'$. Then, the following hold.*

1. $M \oplus L <_c M' \oplus L'$;
2. $M <_c M \oplus L$.[6]

Proof.

1. It is clear that $M \oplus L \leq_c M' \oplus L'$, what remains to prove is thus that I has a winning strategy in $C(M' \oplus L', M \oplus L)$. Let τ be the winning strategy for I in $C(L', L)$. Observe that, since $M \leq_c M'$, player I has a winning strategy τ' in $C(M', M^{\complement})$. A strategy σ for I in the game $C(M' \oplus L', M \oplus L)$ is the following. First I plays 0 on the node ε, and then, as long as player II does not play a 1 on the branch 10^*, I follows τ on the left subtree $0\{0,1\}^*$. If ever II plays a 1 on a node 10^n, then I copies II's moves for the branch 10^n0, and then follows τ' on the subtree $10^n0\{0,1\}^*$. Since τ and τ' are winning, σ is a winning strategy for I in $C(M' \oplus L', M \oplus L)$. Thus $M \oplus L <_c M' \oplus L'$.
2. It is clear that $M \leq_c M \oplus L$: a winning strategy for II in $C(M, M \oplus L)$ is indeed to play 0 at ε, 1 at the node 1, 0 at the node 010, and then copy I's moves in the subtree $010\{0,1\}^*$. The winning strategy σ for I in the game $C(M, M \oplus L)$ is similar. First, I plays 0 at ε, 1 at the node 1, 1 at the node 010, and then copy I's moves in the subtree $010\{0,1\}^*$. ◀

3.2 The pseudo-exponentiation

Let $P \subseteq \mathcal{T}_\Sigma^{\leq\omega}$ be a conciliatory tree language. For $t \in \mathcal{T}_\Sigma^{\leq\omega}$, let:

$$i^P(t)(a_1a_2\dots a_n) = \begin{cases} t(a_10a_20\dots0a_n0) & \text{if } t_{a_10a_20,\dots0a_n1} \in P; \\ b & \text{otherwise.} \end{cases}$$

This process is illustrated in Figure 3. The nodes in red are called the *auxiliary moves*, and the nodes in blue the *main run*. The blue arrows denote the dependency of a node of the main run on a subtree of auxiliary moves. If the auxiliary subtree of a main run node is not in P, then we say that the node is *killed*.

Let $L \subseteq \mathcal{T}_\Sigma^{\leq\omega}$, we define the *action* of P on L, in symbols (P, L), by

$$\left\{t \in \mathcal{T}_\Sigma^{\leq\omega} : i^P(t)_{[\ /b]} \in L\right\}.$$

[6] In particular $M \odot n <_c M \odot (n+1)$ for any $0 < n < \omega$.

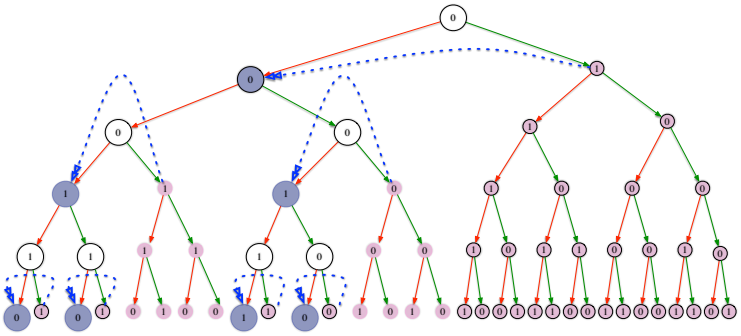


Figure 3

Let $P_{\mathbf{\Pi}^0_1}$ be the complete closed set of all full trees over Σ with all nodes on the leftmost branch 0^* labelled by 0. For $L \subseteq \mathcal{T}_\Sigma^{\leq\omega}$, we denote by $(\mathbf{\Pi}^0_1, L)$ the action of $P_{\mathbf{\Pi}^0_1}$ on L. This operation $(\mathbf{\Pi}^0_1, \cdot)$ behaves well regarding the conciliatory hierarchy.

▶ **Facts 6** ([10, 11]). *Let L and M be conciliatory tree languages over Σ. Then the following hold.*

1. $(\mathbf{\Pi}^0_1, L)^{\mathsf{C}} \equiv_c (\mathbf{\Pi}^0_1, L^{\mathsf{C}})$.
2. *If $L \leq_c M$, then $(\mathbf{\Pi}^0_1, L) \leq_c (\mathbf{\Pi}^0_1, M)$.*
3. *If $L <_c M$, then $(\mathbf{\Pi}^0_1, L) <_c (\mathbf{\Pi}^0_1, M)$.*

The sets obtained as results of the operation $(\mathbf{\Pi}^0_1, \cdot)$ are, so to speak, fixed points for $\oplus$.

▶ **Proposition 7.** *Let L, L' and M be conciliatory languages such that $L <_c (\mathbf{\Pi}^0_1, M)$ and $L' <_c (\mathbf{\Pi}^0_1, M)$. Then*

$$L \oplus L' <_c (\mathbf{\Pi}^0_1, M)$$

Proof. The fact that $L \oplus L' \leq_c (\mathbf{\Pi}^0_1, M)$ is clear: if σ_0, σ_1 and σ' are winning strategies respectively in the games $C(L, (\mathbf{\Pi}^0_1, M))$, $C(L^{\mathsf{C}}, (\mathbf{\Pi}^0_1, M))$ and $C(L', (\mathbf{\Pi}^0_1, M))$, a winning strategy for II in $C(L \oplus L', (\mathbf{\Pi}^0_1, M))$ is the following. As long as player I does not play a 1 on the branch 10^*, II does not kill any nodes and follows σ' to what I plays in the subtree $0\{0,1\}^*$ to get her main run. If ever II plays a 1 on a node 10^n, then II kills all the nodes of the main run she had already played (by playing 1 on the leftmost branches of appropriate auxiliary subtrees), and begins to play along a tree not in M in her main run, without killing any node. If I plays 0 on the node 10^n0, she kills every node in the main run she had already play, and then follows σ_0 on the subtree $10^n0\{0,1\}^*$. If I plays 1 on the node 10^n0, she kills every node in the main run she had already play, and then she follows σ_1 on the subtree $10^n0\{0,1\}^*$. The proof that I has a winning strategy τ in the game $C((\mathbf{\Pi}^0_1, M), L \oplus L')$ is mutatis mutandis the same, given that I has a winning strategy for each of the games $C((\mathbf{\Pi}^0_1, M), L)$, $C((\mathbf{\Pi}^0_1, M), L^{\mathsf{C}})$ and $C((\mathbf{\Pi}^0_1, M), L')$. ◀

Let $\mathcal{A}$ be an automaton that recognizes $L \subseteq \mathcal{T}_\Sigma^{\leq\omega}$. Then the conciliatory tree language $(\mathbf{\Pi}^0_1, L)$ is recognized by the automaton $\boldsymbol{\omega}^{\mathcal{A}}$ defined from $\mathcal{A}$ by replacing each state of $\mathcal{A}$ by a "gadget", as depicted in Figure 4. By replacing a state by the gadget we mean that all transitions ending in this state should now end in the initial state of the gagdet, and that all the transitions starting from this state should now start from the final state of the gadget. This sort of gadget first appeared in [11].

Observe that if $\mathcal{A}$ is unambiguous, then $\boldsymbol{\omega}^{\mathcal{A}}$ is also unambiguous, so that the operation $(\mathbf{\Pi}^0_1, \cdot)$ preserves the unambiguity of conciliatory tree languages.

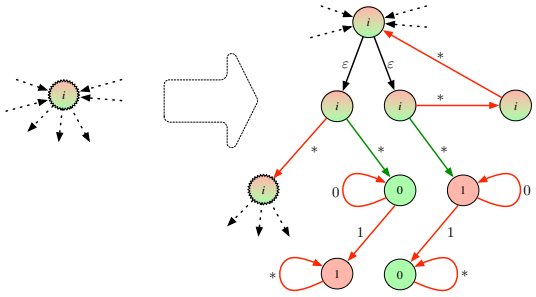
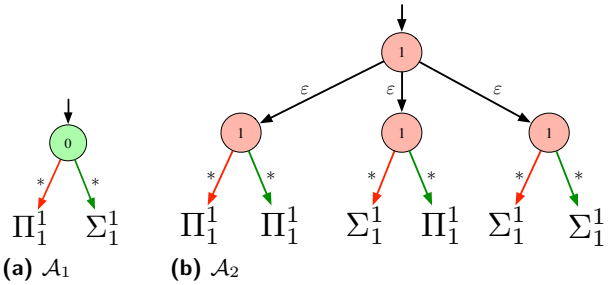


Figure 4 The gadget to replace a state in $\mathcal{A}$.



(a) $\mathcal{A}_1$ **(b)** $\mathcal{A}_2$

Figure 5 Sketch of automata, where $\mathbf{\Pi}^1_1$ and $\mathbf{\Sigma}^1_1$ denote automata that recognize respectively a $\mathbf{\Pi}^1_1$-complete language and the complement of this language.

4 Difference of co-analytic sets

The operations defined in Section 3 are *Borel* in the sense that when we apply them to Borel languages, the resulting language is still Borel. As our purpose is to illustrate the wide discrepancy between deterministic and unambiguously recognizable languages, we need to climb higher in the topological complexity hierarchy. In order to achieve this objective, we will combine a construction due to the third author in [12] with a variant of the pseudo-exponentiation.

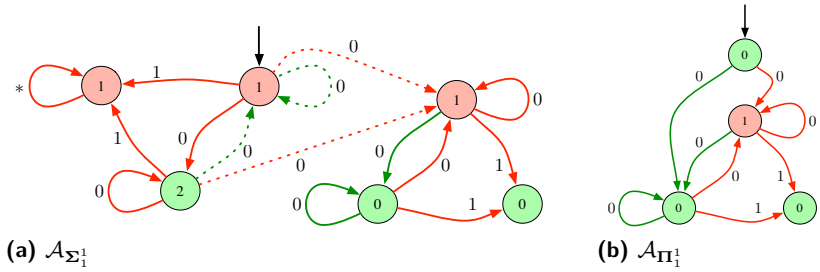
4.1 The $D_2(\mathbf{\Pi}^1_1)$ class

For a topological space X, we denote by $D_2(\mathbf{\Pi}^1_1)(X)$ the class of differences of two coanalytic sets, i.e.

$$D_2(\mathbf{\Pi}^1_1)(X) = \left\{A \cap B : A \in \mathbf{\Pi}^1_1(X) \text{ and } B \in \mathbf{\Sigma}^1_1(X)\right\}.$$

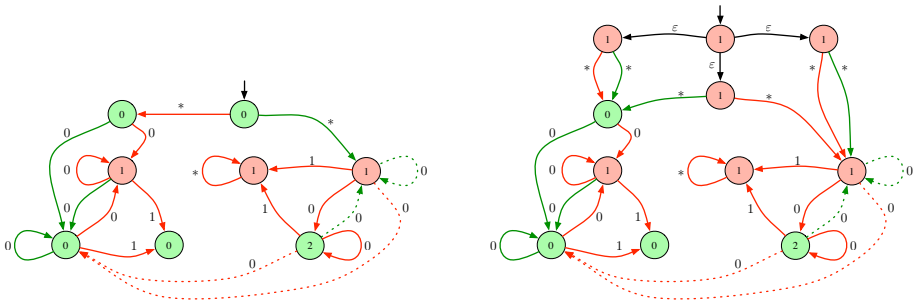
Using the unambiguously recognizable $\mathbf{\Sigma}^1_1$-complete language G (of full trees) from [12], we define an unambiguously recognizable conciliatory language that is $D_2(\mathbf{\Pi}^1_1)$-complete and such that its complement is also unambiguously recognizable. Their definitions are given via the automata that recognize them. The abstract idea behind our construction is depicted by Figure 5 which represents a general form of automata that would recognize languages that are $D_2(\mathbf{\Pi}^1_1)$-complete (Figure 5a), and $\check{D}_2(\mathbf{\Pi}^1_1)$-complete (Figure 5b).

The automaton $\mathcal{A}_1$, indeed, recognizes a tree $t \in \mathcal{T}^{\leq\omega}_{\Sigma}$ if and only if t_0 is in a given conciliatory $\mathbf{\Pi}^1_1$-complete language (say A) and t_1 is in its complement which is $\mathbf{\Sigma}^1_1$-complete. Since the maps $t \mapsto t_0$ and $t \mapsto t_1$ are continuous, the language recognized by the automaton is



(a) $\mathcal{A}_{\mathbf{\Sigma}^1_1}$ (b) $\mathcal{A}_{\mathbf{\Pi}^1_1}$

Figure 6 Unambiguous automata that recognize respectively a $\mathbf{\Sigma}^1_1$-complete language and its complement. Line style of the edges is used on the diagram in case of nondeterministic choice: if there are two different transitions from given state over given letter then egdes of one of them are drawn with solid line while the edges of the other are drawn with dashed line.



(a) $\mathcal{A}_{D_2(\mathbf{\Pi}^1_1)}$ (b) $\mathcal{A}_{\check{D}_2(\mathbf{\Pi}^1_1)}$

Figure 7 Unambiguous automata that recognize respectively a $D_2(\mathbf{\Pi}^1_1)$-complete and a $\check{D}_2(\mathbf{\Pi}^1_1)$-complete language. The transitions that are not depicted lead to an all-accepting state $\top$.

thus $D_2(\mathbf{\Pi}^1_1)$. Moreover, if $M \in \mathbf{\Pi}^1_1$ and $M' \in \mathbf{\Sigma}^1_1$, $M \cap M' \leq_c L(\mathcal{A}_1)$: a winning strategy for player II in the game $C(M \cap M', L(\mathcal{A}_1))$ is indeed to glue together her winning strategies in the games $C(M, A)$ and $C(M', A^{\mathsf{C}})$. Hence, the language recognized by $\mathcal{A}_1$ is $D_2(\mathbf{\Pi}^1_1)$-complete. The reasoning for $\mathcal{A}_2$ is similar, observing that

$$(M \cap M')^{\mathsf{C}} = (M \cap M'^{\mathsf{C}}) \cup (M^{\mathsf{C}} \cap M'^{\mathsf{C}}) \cup (M^{\mathsf{C}} \cap M').$$

We now define two unambiguous automata: the first one recognizes a $\mathbf{\Sigma}^1_1$-complete language, and the other one recognizes the complement of the first one, i.e. a $\mathbf{\Pi}^1_1$-complete language[7]. They are depicted in Figure 6.

We will denote by $A_{\mathbf{\Sigma}^1_1}$ and $A_{\mathbf{\Pi}^1_1}$ the conciliatory languages recognized respectively by $\mathcal{A}_{\mathbf{\Sigma}^1_1}$ and $\mathcal{A}_{\mathbf{\Pi}^1_1}$. Combining these constructions, we can now define an unambiguously recognizable conciliatory language that is $D_2(\mathbf{\Pi}^1_1)$-complete (Figure 7a) and such that its complement (Figure 7b) is also unambiguously recognizable, via the automata that recognize each of them. We will denote by $A_{D_2(\mathbf{\Pi}^1_1)}$ and $A_{\check{D}_2(\mathbf{\Pi}^1_1)}$ the conciliatory languages recognized respectively by $\mathcal{A}_{D_2(\mathbf{\Pi}^1_1)}$ and $\mathcal{A}_{\check{D}_2(\mathbf{\Pi}^1_1)}$.

[7] See [12] for proofs.

4.2 The operation $(D_2(\mathbf{\Pi}^1_1), \cdot)$

For $M \subseteq \mathcal{T}_\Sigma^{\leq\omega}$, we denote by $(D_2(\mathbf{\Pi}^1_1), M)$ the action of $A_{D_2(\mathbf{\Pi}^1_1)}$ on M. Observe that this operation is highly non-Borel, since if we apply it to a $\mathbf{\Sigma}^0_1$-complete conciliatory language, the resulting language will be complete for the pointclass of all the countable unions of $D_2(\mathbf{\Pi}^1_1)$ languages. We prove that $(D_2(\mathbf{\Pi}^1_1), \cdot)$ behaves well with respect to $\leq_c$.

▶ **Theorem 8.** *Let $M, M' \subseteq \mathcal{T}_\Sigma^{\leq\omega}$. If $M \leq_c M'$, then*
1. $(D_2(\mathbf{\Pi}^1_1), M)^{\mathsf{C}} \equiv_c (D_2(\mathbf{\Pi}^1_1), M^{\mathsf{C}})$;
2. $(D_2(\mathbf{\Pi}^1_1), M) \leq_c (D_2(\mathbf{\Pi}^1_1), M')$.

Proof. The first point holds merely by the definition of the operation $(D_2(\mathbf{\Pi}^1_1), \cdot)$. The proof of the second point relies on a variation of the *remote control* strategy ([10]). Let t be a finite binary tree over $\{0,1,2,3\}$. We say that t is *coherent* if for every node $v \in \mathrm{dom}(t)$, $t(v) \in \{1,2,3\}$ implies that all the nodes in $v1\{0,1\}^* \cap \mathrm{dom}(t)$ have the same label, $t(v)$. Let $(\beta_n)_{n\in\omega}$ be an enumeration of the set of the coherent trees, such that if t_i is an initial segment of t_j, then $i \leq j$. We call β_i the i-th *bet*. A bet encodes information on the auxiliary moves of I in the game $C((D_2(\mathbf{\Pi}^1_1), M), (D_2(\mathbf{\Pi}^1_1), M'))$: its underlying binary tree determines the part of the main run taken into account, and the values at the nodes whether this node will be killed or not, and how. Suppose I plays a conciliatory tree t. For $v = v_0 \dots v_j \in \mathrm{dom}(\beta_i)$, $\beta_i(v) = 0$ means that the node $0v_00v_1 \dots 0v_j$ stays alive, i.e. that $t_{0v_00v_1\dots0v_j1} \in A_{D_2(\mathbf{\Pi}^1_1)}$. The value 1 means that the node $0v_00v_1 \dots 0v_j$ is killed because $t_{0v_00v_1\dots0v_j10}$ and $t_{0v_00v_1\dots0v_j11}$ belong to $A_{\mathbf{\Pi}^1_1}$, so that $t_{0v_00v_1\dots0v_j1} \in A_{\check{D}_2(\mathbf{\Pi}^1_1)}$. The value 2 means that the node $0v_00v_1 \dots 0v_j$ is killed because $t_{0v_00v_1\dots0v_j10} \in A_{\mathbf{\Sigma}^1_1}$ and $t_{0v_00v_1\dots0v_j11} \in A_{\mathbf{\Pi}^1_1}$, and the value 3 means that it is killed because both $t_{0v_00v_1\dots0v_j10}$ and $t_{0v_00v_1\dots0v_j11}$ belong to $A_{\mathbf{\Sigma}^1_1}$. We say that a bet β_i is fulfilled if at the end of the game, for all $v \in \mathrm{dom}(\beta_i)$, $\beta_i(v)$ is true with respect to the conciliatory tree played by I. Notice that it is a $D_2(\mathbf{\Pi}^1_1)$ condition (it is a finite intersection of $\mathbf{\Sigma}^1_1$ and $\mathbf{\Pi}^1_1$ sets), so that II can check if a bet is fulfilled or not with an auxiliary move.

Suppose now that II has a winning strategy σ in $C(M, M')$. We describe a winning strategy σ' for II in the game $C((D_2(\mathbf{\Pi}^1_1), M), (D_2(\mathbf{\Pi}^1_1), M'))$. Each level of II's main run corresponds to a bet: suppose at some point I has constructed a finite tree t for his main run, and let β_i be a bet such that $\mathrm{dom}(t) = \mathrm{dom}(\beta_i)$. On the level i of her main run, II follows σ modulo β_i, in the sense that she plays along σ as if at all the levels $j < i$ of her main run such that β_j is not a subtree of β_i, the nodes were killed, and she checks with her auxiliary moves for the nodes of the main run at this level whether β_i is fulfilled or not, so that all the nodes of her main run at this level are killed if the bet is not fulfilled. At the end of the game, a unique sequence of bets forming a chain for the inclusion is fulfilled, which contains all information about the way player I used his auxiliary moves, and which nodes he killed. Hence,

$$i^{A_{D_2(\mathbf{\Pi}^1_1)}}(\sigma' * t)_{[\ /b]} = \sigma * i^{A_{D_2(\mathbf{\Pi}^1_1)}}(t)_{[\ /b]},$$

where $\sigma * t$ denotes the tree resulting from application of strategy σ to tree t. That finishes the proof. ◀

Mutatis mutandis, a winning strategy for I in $C(M, M')$ can also be "remote controlled" to a winning strategy for I in $C((D_2(\mathbf{\Pi}^1_1), M), (D_2(\mathbf{\Pi}^1_1), M'))$.

▶ **Corollary 9.** *Let M and M' be conciliatory languages such that $M <_c M'$. Then*

$$(D_2(\mathbf{\Pi}^1_1), M) <_c (D_2(\mathbf{\Pi}^1_1), M')$$

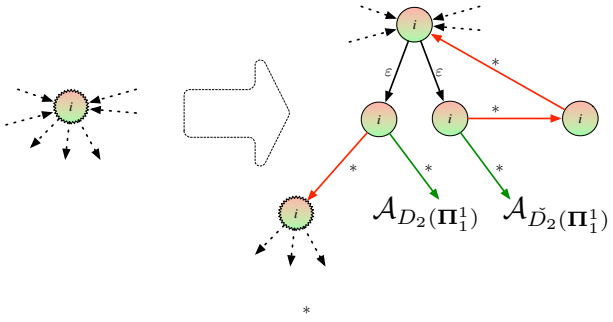


Figure 8 The gadget to replace a state in $\mathcal{A}$. The transitions that are not depicted lead to an all-accepting state $\top$.

The operation $(D_2(\mathbf{\Pi}^1_1), \cdot)$ is much stronger than $(\mathbf{\Pi}^0_1, \cdot)$, and is in fact a fixpoint of it.

▶ **Proposition 10.** *Let $M \subseteq \mathcal{T}_\Sigma^{\leq\omega}$. Then*

$$\big(\mathbf{\Pi}^0_1, (D_2(\mathbf{\Pi}^1_1), M)\big) \equiv_c (D_2(\mathbf{\Pi}^1_1), M).$$

The proof of Proposition 10 is a variant of the remote-control technique and is omitted here.

Let $\mathcal{A}$ be an automaton that recognizes $M \subseteq \mathcal{T}_\Sigma^{\leq\omega}$. Then the conciliatory tree language $(D_2(\mathbf{\Pi}^1_1), M)$ is recognized by the automaton $\varepsilon_\mathcal{A}$ defined from $\mathcal{A}$ by replacing each state of $\mathcal{A}$ by a "gadget", as depicted in Figure 8. As in the pseudo-exponentiation case, by replacing a state by the gadget we mean that all transitions ending in this state should now end in the initial state of the gagdet, and that all the transitions starting from this state should now start from the final state of the gadget.

Observe that, since $A_{D_2(\mathbf{\Pi}^1_1)}$ and $A_{\check{D}_2(\mathbf{\Pi}^1_1)}$ are disjoint, if $\mathcal{A}$ is unambiguous, then $\varepsilon_\mathcal{A}$ is also unambiguous, so that the operation $(D_2(\mathbf{\Pi}^1_1), \cdot)$ preserves the unambiguity of conciliatory tree languages.

5 A fragment of the unambiguous Wadge hierarchy

Consider the *epsilon function*, the ordinal function that enumerates the fixed-points of the exponentiation of base ω:

$$\varepsilon_0 = \sup_{n<\omega} \underbrace{\omega^{\cdot^{\cdot^{\omega^0}}}}_{n}; \qquad \varepsilon_{\alpha+1} = \sup_{n<\omega} \underbrace{\omega^{\cdot^{\cdot^{\omega^{(\varepsilon_\alpha+1)}}}}}_{n}; \qquad \varepsilon_\lambda = \sup_{\alpha<\lambda} \varepsilon_\alpha \text{ for } \lambda \text{ limit.}$$

We denote by $\varphi_2(0)$ its first fixed-point:

$$\varphi_2(0) = \sup_{n<\omega} \underbrace{\varepsilon_{\cdot^{\cdot^{\varepsilon_0}}}}_{n}.$$

The ordinal $\varphi_2(0)$ is the first value of the second function of the Veblen hierarchy [17]. Another way to characterise $\varphi_2(0)$ is to remember that an ordinal is the set of its predecessors and notice that a non-zero ordinal is of the form respectively ω^α if and only if it is closed under addition, and ε_α if and only if it is closed under $x \longmapsto \omega^x$. Then $\varphi_2(0)$ is the first non-zero ordinal closed under $x \longmapsto \varepsilon_x$ as well as $x \longmapsto \omega^x$ and $x, y \longmapsto x + y$.

We recall that every ordinal $\alpha > 0$ admits a unique Cantor normal form of base ω (CNF) which is an expression of the form

$$\alpha = \omega^{\alpha_k} \cdot n_k + \cdots + \omega^{\alpha_0} \cdot n_0$$

where $k < \omega$, $0 < n_i < \omega$ (any $i \leq k$) and $\alpha_0 < \cdots < \alpha_k < \alpha$.

For every ordinal $0 < \alpha < \varphi_2(0)$, we inductively define a pair of unambiguous automata $(\mathcal{A}_\alpha, \bar{\mathcal{A}}_\alpha)$ whose languages are both non-selfdual and incomparable through the conciliatory ordering. If the CNF of α is $\alpha = \omega^{\alpha_k} \cdot n_k + \cdots + \omega^{\alpha_0} \cdot n_0$ we set

$$\mathcal{A}_\alpha = \mathcal{A}_{\omega^{\alpha_k}} \bullet n_k + \cdots + \mathcal{A}_{\omega^{\alpha_0}} \bullet n_0$$

and

$$\bar{\mathcal{A}}_\alpha = \mathcal{A}_{\omega^{\alpha_k}} \bullet n_k + \cdots + \bar{\mathcal{A}}_{\omega^{\alpha_0}} \bullet n_0,$$

where $\mathcal{A}_{\omega^{\alpha_i}}$ and $\bar{\mathcal{A}}_{\omega^{\alpha_i}}$ are respectively:

- $\ominus$ and $\oplus$ if $\alpha_i = 0$;
- $\boldsymbol{\omega}^{\mathcal{A}_{\alpha_i}}$ and $\boldsymbol{\omega}^{\bar{\mathcal{A}}_{\alpha_i}}$ if $\alpha_i < \omega^{\alpha_i}$;
- $\boldsymbol{\varepsilon}_{\mathcal{A}_{2+\beta}}$ and $\boldsymbol{\varepsilon}_{\bar{\mathcal{A}}_{2+\beta}}$ if $\alpha_i = \omega^{\alpha_i}$ and $\alpha_i = \varepsilon_\beta$ for some $\beta < \alpha_i$.

Here $\oplus$ denotes automaton that accepts all conciliatory trees.

▶ **Lemma 11.** *Let* $0 < \alpha < \beta < \varphi_2(0)$,

1. $\mathcal{A}_\alpha \not\leq_c \bar{\mathcal{A}}_\alpha$ *and* $\bar{\mathcal{A}}_\alpha \not\leq_c \mathcal{A}_\alpha$.
2. $\mathcal{A}_\alpha <_c \mathcal{A}_\beta$; $\bar{\mathcal{A}}_\alpha <_c \mathcal{A}_\beta$; $\mathcal{A}_\alpha <_c \bar{\mathcal{A}}_\beta$ *and* $\bar{\mathcal{A}}_\alpha <_c \bar{\mathcal{A}}_\beta$.

Proof.

1. The proof, by induction on α, relies on the fact that the operations considered "commute" with each others, see Facts 4, 6 and Theorem 8.
2. The proof, by induction on α and β, relies on the fact that the operations preserve the relation $<_c$ (see Lemma 5, Facts 6 and Corollary 9) on the one hand, and on the fact that they do not "overlap" (see Propositions 7 and 10). ◀

Applying the embedding $L \mapsto L^b$, we have thus generated a family $\left(\mathcal{A}^b_\alpha\right)_{\alpha<\varphi_2(0)}$ of unambiguous automata that respects the strict Wadge ordering: $\alpha < \beta$ if and only if $\mathcal{A}^b_\alpha <_W \mathcal{A}^b_\beta$. Even though the exact Wadge rank of this family is unknown, this fragment of the $\mathbf{\Delta}^1_2$-Wadge hierarchy restricted to unambiguously recognizable languages climbs far above the $\mathbf{\Sigma}^1_1$ class. Hence the main result follows.

▶ **Theorem 12.** *There exists a family* $\left(\mathcal{A}^b_\alpha\right)_{\alpha<\varphi_2(0)}$ *of unambiguous parity tree automata whose priorities are restricted to* $\{0,1,2\}$ *such that*

1. *they recognize languages of full trees over the alphabet* $\{0,1,b\}$;
2. $\alpha < \beta$ *holds if and only if* $\mathcal{A}^b_\alpha <_W \mathcal{A}^b_\beta$ *holds as well.*

6 Conclusion

In this paper, we have produced a very long chain of unambiguous parity tree automata of different Wadge degrees. Its length, the ordinal $\varphi_2(0)$, is the first fixpoint of the ordinal function that itself enumerates all fixpoints of the ordinal exponentiation $x \mapsto \omega^x$. All these automata share a Rabin-Mostowski index of at most $(0,2)$. This indicates that the whole Wadge hierarchy of unambiguous parity tree automata is even far more complicated than that, not to mention the even higher complexity of the Wadge hierarchy of regular tree

languages which, in comparison, seems scary. This illustrates in particular how different the tree-case scenario is from the word-case scenario.

The whole construction is effective. This means that the mapping $\alpha \mapsto \mathcal{A}^b_\alpha$ (for $0 < \alpha < \varphi_2(0)$) is recursive. And also that, for any $0 < \alpha < \beta < \varphi_2(0)$, the relation $\mathcal{A}^b_\alpha <_W \mathcal{A}^b_\beta$ which stipulates that there exists two strategies – one that is winning for player II in the game $W(\mathcal{A}^b_\alpha, \mathcal{A}^b_\beta)$ and another one that is winning for I in the game $W(\mathcal{A}^b_\beta, \mathcal{A}^b_\alpha)$ – can be established by recursively providing such strategies.

However, we did not consider any decidability issue. It thus remains open to show whether one can decide, given any automaton $\mathcal{B}$ and any ordinal $0 < \alpha < \varphi_2(0)$, whether $\mathcal{B} <_W \mathcal{A}^b_\alpha$ holds or not.

References

1 Alessandro Andretta and Alain Louveau. Wadge degrees and pointclasses. In Alexander S. Kechris, Benedikt Löwe, and John R. Steel, editors, *Wadge Degrees and Projective Ordinals: The Cabal Seminar, Volume II*. Cambridge University Press, 2012.

2 André Arnold. Rational omega-languages are non-ambiguous. *Theoretical Computer Science*, 26:221–223, 1983.

3 André Arnold, Jacques Duparc, Filip Murlak, and Damian Niwiński. On the topological complexity of tree languages. *Logic and automata: History and Perspectives*, 2:9–29, 2007.

4 Mikołaj Bojańczyk. A bounding quantifier. In *CSL*, pages 41–55, 2004.

5 Mikołaj Bojańczyk, Tomasz Gogacz, Henryk Michalewski, and Michał Skrzypczak. On the decidability of MSO+U on infinite trees. In *ICALP*, pages 50–61, 2014.

6 Mikołaj Bojańczyk, Paweł Parys, and Szymon Toruńczyk. The MSO+U theory of (N, <) is undecidable. *CoRR*, abs/1502.04578, 2015.

7 Arnaud Carayol and Christof Löding. MSO on the infinite binary tree: Choice and order. In *CSL*, pages 161–176, 2007.

8 Arnaud Carayol, Christof Löding, Damian Niwiński, and Igor Walukiewicz. Choice functions and well-orderings over the infinite binary tree. *Central European Journal of Mathematics*, 8:662–682, 2010.

9 Thomas Colcombet. Forms of determinism for automata (invited talk). In *STACS*, pages 1–23, 2012.

10 Jacques Duparc. Wadge hierarchy and Veblen hierarchy, Part I : Borel sets of finite rank. *The Journal of Symbolic Logic*, 66(1):56–86, 2001.

11 Jacques Duparc and Filip Murlak. On the topological complexity of weakly recognizable tree languages. In *Fundamentals of Computation Theory, 16th International Symposium, FCT 2007, Budapest, Hungary, August 27-30, 2007, Proceedings*, pages 261–273, 2007.

12 Szczepan Hummel. Unambiguous tree languages are topologically harder than deterministic ones. In *GandALF*, pages 247–260, 2012.

13 Filip Murlak. The Wadge hierarchy of deterministic tree languages. *Logical Methods in Computer Science*, 4(4), 2008.

14 Damian Niwiński and Igor Walukiewicz. Ambiguity problem for automata on infinite trees. unpublished note, 1996.

15 Michael O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the AMS*, 141:1–23, 1969.

16 Robert Van Wesep. Wadge degrees and descriptive set theory. In Alexander S. Kechris, Benedikt Löwe, and John R. Steel, editors, *Wadge Degrees and Projective Ordinals: The Cabal Seminar, Volume II*. Cambridge University Press, 2012.

17 Oswald Veblen. Increasing functions of finite and transfinite ordinals. *Transactions of the American Mathematical Society*, 9(3):280–292, 1908.

18 William W. Wadge. *Reducibility and determinateness on the Baire space*. PhD thesis, University of California, Berkeley, 1984.

Least and Greatest Fixed Points in Ludics

David Baelde[1], Amina Doumane[2], and Alexis Saurin[2]

1 **LSV, ENS Cachan, France**
baelde@lsv.ens-cachan.fr
2 **PPS, CNRS, Université Paris Diderot & Inria, France**
{amina.doumane,alexis.saurin}@pps.univ-paris-diderot.fr

Abstract

Various logics have been introduced in order to reason over (co)inductive specifications and, through the Curry-Howard correspondence, to study computation over inductive and coinductive data. The logic μMALL is one of those logics, extending multiplicative and additive linear logic with least and greatest fixed point operators.

In this paper, we investigate the semantics of μMALL proofs in (computational) ludics. This framework is built around the notion of design, which can be seen as an analogue of the strategies of game semantics. The infinitary nature of designs makes them particularly well suited for representing computations over infinite data. We provide μMALL with a denotational semantics, interpreting proofs by designs and formulas by particular sets of designs called behaviours. Then we prove a completeness result for the class of "essentially finite designs", which are those designs performing a finite computation followed by a copycat. On the way to completeness, we establish decidability and completeness of semantic inclusion.

1998 ACM Subject Classification F.1.1 Models of Computation, F.4.1 Mathematical Logic

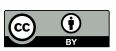
Keywords and phrases proof theory, fixed points, linear logic, ludics, game semantics, completeness, circular proofs, infinitary proof systems

Digital Object Identifier 10.4230/LIPIcs.CSL.2015.549

1 Introduction

Through the Curry-Howard correspondence, proof theory allows to design and study programming languages in which programs are correct by construction: formulas correspond to types and proofs correspond to well-typed, pure and total programs. Like programs, proofs generally contain irrelevant information which may obfuscate their computational contents, making it hard to tell when two proofs correspond to the same computation, or to characterize the class of computations being expressible as proofs. A major goal of proof theory is to tackle these problems, by identifying and eliminating such syntactic noise to get down to the essence of proofs.

Following this tradition, the proof theory of least and greatest fixed points provides a way to design and study programming constructs associated to inductive and coinductive types. Such types can be encoded using second-order quantification, *e.g.*, $\forall X.\ X \to (X \to X) \to X$ represents natural numbers through primitive recursion. However, the encoding has several undesirable effects: it notably forces impredicativity into the system, and results in indirect ways of computing over (co)inductive objects. These two reasons have motivated the introduction of fixed points in type theory. Mendler [17] and Matthes [15] extended second-order λ-calculus with least and greatest fixed point types and an associated generic primitive recursion operator. Similar developments took place in richer type theories, supporting the introduction of inductive and coinductive specifications in tools such as Coq or Agda.



24th EACSL Annual Conference on Computer Science Logic (CSL 2015).
Editor: Stephan Kreutzer; pp. 549–566

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

However, these aspects are far from being fully understood: there is a history of bugs pertaining to the guard condition that restricts (co)recursion to ensure totality in these systems. This has spurred the development of other approaches to (co)induction, such as sized types [4] and (co)patterns [1]. The works cited above are mostly concerned with strong normalization, and do not investigate the computational content of proofs beyond this property. Various other works, generally taking place in weaker logics, have investigated more closely the semantics of proof on (co)inductive types, notably by means of infinite proofs and games [19, 11, 8, 9, 7].

In this paper, we carry out a similar semantical investigation, revealing the computational content of proofs by means of an infinitary semantics, and tackling the difficult problem of completeness for this semantical interpretation. We work with the system μMALL [2], which extends multiplicative and additive linear logic[1] with least and greatest fixed point operators. In addition to simple fixed point unfolding rules, μMALL features, like most of the systems cited above, generic primitive (co)recursion rules shown below in a simplified two-sided form:

$$\frac{F[S/X] \vdash S}{\mu X.F \vdash S}\ (\mu_L) \qquad \frac{S \vdash F[S/X]}{S \vdash \nu X.F}\ (\nu_R)$$

The induction principle (μ_L) behaves in cut elimination as a recursive process, transforming any derivation of $\vdash \mu X.F$ into a derivation of $\vdash S$. Now, we can express in that framework the type of lists of A as $L = \mu X.1 \oplus (A \otimes X)$ and the type of infinite streams of A as $S = \nu X.A \otimes X$. We can then define a proof (shown next) that concatenates a list and a stream into a stream, by recursing over the list with the invariant $S \multimap S$. It is not trivial to convince oneself that this proof does compute the concantenation function. More generally, it is hard to tell what such proofs compute, when two proofs compute the same function, etc. It requires to step back from the finite, syntactic proof system under consideration and to start considering its semantics; this is the topic of the present paper.

$$\dfrac{\dfrac{\dfrac{}{1 \vdash S \multimap S}\ (\dots, \mathrm{ax}) \qquad \dfrac{\dfrac{\dfrac{}{A, S \multimap S, S \vdash A \otimes S}\ (\dots, \mathrm{ax})}{A, S \multimap S, S \vdash S}\ (\mathrm{unfold}_R)}{A \otimes (S \multimap S) \vdash S \multimap S}\ (\otimes_L, \multimap R)}{1 \oplus (A \otimes (S \multimap S)) \vdash S \multimap S}\ (\oplus_L)}{L \vdash S \multimap S}\ (\mu_L)$$

As our domain of interpretation of proofs, we consider ludics [12] which can be regarded as a variant of game semantics, where the basic objects are well-behaved strategies, called designs. Girard introduced ludics with the aim of bringing closer together syntax and semantics in the study of proofs and proved a full completeness result with respect to proofs of a polarized variant of MALL. Extending this interpretation to all of linear logic, including exponentials, has been challenging and required to deal with non-determinism [5]. As we shall see, accounting for least and greatest fixed points is much easier, and can essentially be done in Girard's original framework. Still, we shall work in Terui's reformulation of ludics, *computational ludics* [20], since it is more convenient to work with and slightly more general; for instance the objects of computational ludics may contain cut while Girard's original designs are cut-free: this happens to be very handy when working with greatest fixed point.

Contributions. Our first contribution is a denotational semantics for (a polarized version of) μMALL: we interpret formulas as well-behaved sets of designs, and proofs as designs belonging to the conclusion sequent's interpretation. Our second contribution is a completeness result

[1] While linearity is certainly a restriction, it is not a severe one with respect to our main interest in this paper. Note moreover that μMALL is already very expressive as it contains at least all primitive recursive functions.

for a class of designs we call *essentially finite designs* (EFD) which are designs performing a finite computation followed by a copycat. To prove this result, we investigate semantic inclusion, proving its decidability (*i.e.*, whether the intepretations of two formulas are included in each other) and completeness (if semantic inclusion holds, the corresponding entailment is μMALL-provable). This last result relies on a circular proof system, in the style of Santocanale [19], as a stepping stone between infinite designs and finite proofs.

Outline. We introduce μMALLP (the polarized variant of μMALL) in Section 2 followed by ludics in Section 3. We then define the interpretation of μMALLP proofs in ludics in Section 4, and prove its soundness. Finally, we establish completeness for EFD in Section 5. Detailed proofs may be found in the long version of this paper [3].

2 Linear Logic with Fixed Points

In this section, we formally introduce our logic with least and greatest fixed points. As usual when aiming at ludics interpretation [12, 5, 20] we will actually be working with a polarized version of μMALL [2], μMALLP, to which the present section is dedicated.

We assume two infinite and disjoint sets $\mathcal{V}_P$ and $\mathcal{V}_N$, whose elements are respectively called positive and negative variables and denoted by X_P and X_N, or simply X when their polarity is irrelevant or can be inferred from the context.

▶ **Definition 1.** The sets of *positive preformulas* $P, Q, \dots$ and of *negative preformulas* $N, M, \dots$ are inductively defined by the following grammar:

$$\begin{array}{rcl} P,\ Q & ::= & X_P \mid X_N^\perp \mid 1 \mid 0 \mid N \oplus M \mid N \otimes M \mid \downarrow N \mid \mu X_P.P \\ N, M & ::= & X_N \mid X_P^\perp \mid \bot \mid \top \mid P \,\&\, Q \mid P \parr Q \mid \uparrow P \mid \nu X_N.N \end{array}$$

The connectives μ and ν are variable binders, and the notions of free and bound variables are as usual. *Formulas* are those preformulas with no free variables. A preformula is said to be *monotonic* if it contains no negated variable $X^\perp$, neither free nor bound. A formula is said to be *degenerate* if it contains either $\mu X.X$ or $\nu X.X$ as a subformula.

Nested fixed points correspond to iterated (co)inductive definitions. For example, $\mathrm{Nat} := \mu X.\ (\uparrow 1) \oplus (\uparrow X)$ is the type of natural numbers, and $\nu Y.\ \uparrow((\uparrow \mathrm{Nat}) \otimes Y)$ is the type of infinite streams of natural numbers. Fixed points can also be interleaved, which corresponds to mutual (co)inductive definitions. For example, $\mu X.\ T \otimes (\nu Y.\ \uparrow((\uparrow 1) \oplus ((\uparrow X) \otimes Y)))$ is the type of arbitrarily branching well-founded trees, with data of type T as every node – such trees have no infinite branch, but each node may have infinitely many children.

Our syntax classifies μ as positive and ν as negative. This is a natural choice but it is not forced: all of this work could be done by taking the opposite classification, which is consistent with the observations made in the study of focusing for μMALL [2]. In a nutshell, the polarity of a fixed point formula is not forced by the fixed point operator but rather by the formula inside the fixed point. In that setting, the formulas $\mu X.X$ and $\nu X.X$ have no meaningful polarity. We shall thus assume from now on that all formulas are non-degenerate.

▶ **Definition 2.** *Negation* is the involutive operation mapping positive to negative preformulas, and vice versa, such that:

$$(F_1 \parr F_2)^\perp = F_1^\perp \otimes F_2^\perp \quad (F_1 \,\&\, F_2)^\perp = F_1^\perp \oplus F_2^\perp \quad (\uparrow F)^\perp = \downarrow F^\perp$$
$$(\nu X.F)^\perp = \mu Y.(F^\perp[Y^\perp/X]) \quad (X_N)^\perp = X_N^\perp \quad (X_P^\perp)^\perp = X_P \quad \top^\perp = 0 \quad \bot^\perp = 1$$

Identity rules:

$$\frac{}{\vdash P, P^\perp}\ (\text{ax}) \qquad \frac{\vdash \Gamma, P^\perp \quad \vdash \Delta, P}{\vdash \Gamma, \Delta}\ (\text{cut})$$

Fixed point rules:

$$\frac{\vdash \Gamma, P[\mu X.P/X]}{\vdash \Gamma, \mu X.P}\ (\mu) \qquad \frac{\vdash \Gamma, S \quad \vdash S^\perp, N[S/X]}{\vdash \Gamma, \nu X.N}\ (\nu)$$

MALL rules:

$$\frac{}{\vdash \Gamma, \top}\ (\top) \qquad \frac{\vdash \Gamma}{\vdash \Gamma, \bot}\ (\bot) \qquad \frac{}{\vdash 1}\ (1) \qquad \frac{\vdash \Gamma, P}{\vdash \Gamma, \uparrow P}\ (\uparrow) \qquad \frac{\vdash \Gamma, N}{\vdash \Gamma, \downarrow N}\ (\downarrow)$$

$$\frac{\vdash \Gamma, P_1 \quad \vdash \Gamma, P_2}{\vdash \Gamma, P_1 \& P_2}\ (\&) \qquad \frac{\vdash \Gamma, N_i}{\vdash \Gamma, N_1 \oplus N_2}\ (\oplus_i) \qquad \frac{\vdash \Gamma, P_1, P_2}{\vdash \Gamma, P_1 \parr P_2}\ (\parr) \qquad \frac{\vdash \Delta, N_1 \quad \vdash \Gamma, N_2}{\vdash \Gamma, \Delta, N_1 \otimes N_2}\ (\otimes)$$

In these rules, Γ and Δ denote positive sequents, *i.e.*, ones that contain only positive formulas.

Figure 1 The μMALLP sequent calculus proof system.

From now on, we restrict our attention to monotonic (pre)formulas. This natural restriction rules out formulas such as $\mu X.{\downarrow}X^\perp$, which would yield inconsistencies. Assuming monotonicity amounts to fully remove negation from our syntax — the presence of negated variables in it is only useful to be able to define negation. We may still use negation as an operation on formulas, since it preserves monotonicity.

We denote by $F[\vec{G}/\vec{X}]$ the preformula obtained by the simultaneous capture-avoiding substitution of the variables $\vec{X}$ by the preformulas $\vec{G}$. When considering a substitution $F[\vec{G}/\vec{X}]$, we always assume implicitly that the polarities of $\vec{G}$ are adequate to those of $\vec{X}$.

▶ **Definition 3.** The proof system μMALLP is given in Figure 1. It is a focused sequent calculus over our polarized syntax, meaning that its sequents must contain at most one negative formula. A sequent is said to be *negative* when it contains a negative formula, and it is *positive* otherwise. In Figure 1, Γ and Δ always denote positive sequents.

Reading proofs in a proof search (bottom-up) fashion, the polarity restriction on sequents means that negative rules must be applied eagerly, *i.e.*, as soon as the sequent contains a negative formula. This constraint on the shape of proofs is a very mild form of focusing. Note that we can simply translate between μMALL and μMALLP, in the same way as is done between MALL and MALLP: μMALL formulas are translated into μMALLP formulas by inserting shift connectives, and any μMALL proof can be turned into a μMALLP proof of the translated conclusion sequent by inserting shift rules[2]; in the other direction, shifts are simply erased.

As mentioned in the introduction, the fixed point rules of μMALLP can be understood from Knaster-Tarski's characterization of an operator's extremal fixed points in complete lattices. Rule (μ) expresses that $\mu X.P$ is a pre-fixed point of $X \mapsto P$, provided that one reads implication as inclusion ($P[\mu X.P/X] \multimap \mu X.P$). Similarly, we may express that the greatest fixed point is greater than all post-fixed points by the following rule:

$$\frac{S \vdash N[S/X]}{S \vdash \nu X.N}\ (\nu_0)$$

[2] This translation essentially cancels the focusing constraint of the μMALLP proof system by inserting shifts. A more demanding task would be to establish completeness of the focused μMALLP proof system given here with respect to an unfocused proof system for μMALLP. We do not address this (unrelated) issue, but expect that it would be possible along the lines of the focusing result for μMALL [2].

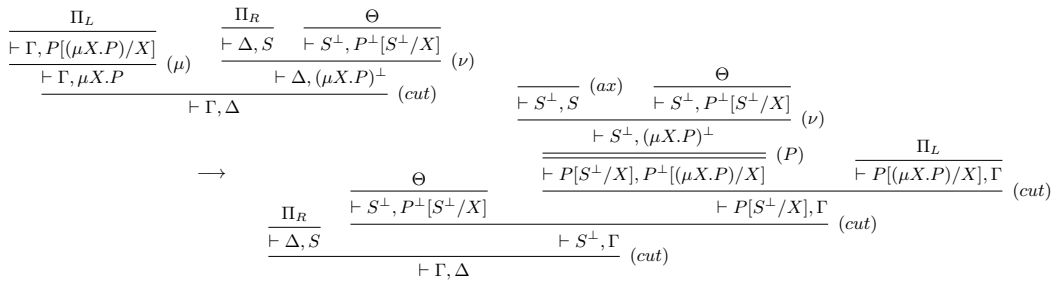


Figure 2 (μ)–(ν) Key cut-elimination step.

Rule (ν) of Figure 1 is obtained from this one by combining it with a cut against the co-invariant S – this presentation is preferred because it yields a system that enjoys cut elimination.

The above explanation may be helpful to understand the rules at first, but it does not say anything about their computational interpretation. Cut elimination holds for μMALLP: the cut reduction system given in [2] can straightforwardly be adapted to the polarized setting of μMALLP. As the reader may expect, the only specific case is the one involving least and greatest fixed points. This cut reduction step, shown in Figure 2, relies on the *functoriality* construction: if Π is a proof of a sequent $\vdash P, N$ then $F_G(\Pi)$ is a particular proof of $\vdash G[P/X], G^\perp[N^\perp/X]$, whose precise definition may be found in [2]. Operationally, functoriality should be viewed as a `map` operator in functional programming. The role of $F_G(\Pi)$ is to apply Π to all occurrences of X in G. Roughly, its type may be read as $(N^\perp \to P) \to (G[N^\perp/X] \to G[P/X])$. In the cut reduction step of Figure 2, this has the effect of propagating the (ν) rule to the next occurrences of $(\mu X.P)^\perp$ in $P^\perp[(\mu X.P)^\perp/X]$. Overall, the reduction achieves in this way the (co)recursive computational behaviour described in the introduction[3]. Intuitively, this process terminates because each application of this reduction consumes a (μ) rule; see [2] for the formal argument.

3 Computational Ludics

Ludics is an interactive framework reminiscent from and somehow intermediate between game semantics [13] and realizability [14]. We recall, in the setting of computational ludics [20], the necessary definitions and properties of

- designs (§ 3.1), which correspond to strategies,
- orthogonality (§ 3.2), which corresponds to interaction, and
- behaviours (§ 3.3), which correspond to arenas or interactive types.

In game semantics, arenas are defined first and strategies are defined as sets of plays (or as sets of views) compatible with these arenas. However, in ludics, arenas are a secondary notion, derived from that of designs since behaviours are obtained from designs by orthogonality; we develop this comparison in § 3.4.

[3] Note that in our classical linear logic setting, induction and coinduction (in other words, recursion and corecursion) become the same.

3.1 Designs

Designs are built over a *signature* $\mathcal{A} = (A, \mathrm{ar})$, where A is a set of names $a, b, c, \dots$ and $\mathrm{ar} : A \to \mathbb{N}$ is a function which assigns to each name a its *arity* $\mathrm{ar}(a)$. Let V be a set of variables $V = \{x, y, z, \dots\}$.

▶ **Definition 4.** For a fixed signature $\mathcal{A}$, the class of *positive designs* $p, q, \dots$ and *negative designs* $n, m, \dots$ are coinductively defined as follows (with $\mathrm{ar}(a) = \mathrm{card}(\vec{x}_a) = k$):

$$p \ ::= \quad \Omega \quad | \quad \maltese \quad | \quad (n_0 \mid \overline{a}\langle n_1, \dots, n_k\rangle) \qquad n \ ::= \quad x \quad | \quad \sum a(\vec{x}_a).p_a$$

The formal sum $\sum a(\vec{x}_a).p_a$ is the $\mathcal{A}$-indexed family $\{a(\vec{x}_a).p_a\}_{a\in\mathcal{A}}$.

We write $\sum_{K\subseteq\mathcal{A}} a(\vec{x}_a).p_a$ to denote the *design* $\sum a(\vec{x}_a).q_a$ where $q_a = p_a$ if $a \in K$ and $q_a = \Omega$ otherwise. We denote by Ω^- the design $\sum a(\vec{x}_a).\Omega$. In a negative design $\sum a(\vec{x}_a).p_a$, $a(\vec{x}_a)$ binds the variables $\vec{x}_a$ appearing in p_a. Variables which are not under the scope of a binder are free. The free variables of a design d are denoted by $\mathrm{fv}(d)$. We identify two designs which are α-equivalent, *i.e.*, which are equal up to renaming of their bound variables. We denote by $d[\vec{n}/\vec{x}]$ the design obtained by a simultaneous and capture-free substitution of the variables $\vec{x}$ by the negative designs $\vec{n}$. The reader is referred to [20] for precise definitions.

▶ **Definition 5** (l-designs, standard designs)**.** A design of the form $n_0 \mid \overline{a}\langle n_1, \dots, n_k\rangle$ where n_0 is not a variable is called a *cut*. An occurrence of a variable x is called an *identity* if it occurs as $n_0 \mid \overline{a}\langle n_1, \dots, x, \dots, n_k\rangle$. We call a design *identity-free* (resp. *cut-free*) if it does not contain an identity (resp. a cut) as a subdesign. A design d is called *linear* if for every positive subdesign $n_0 \mid \overline{a}\langle n_1, \dots, n_k\rangle$ of d, the sets $\mathrm{fv}(n_0), \dots, \mathrm{fv}(n_k)$ are pairwise disjoint. An *l-design* is a design d which is linear, identity-free and such that $\mathrm{fv}(d)$ is finite. A *standard* design is a cut-free l-design.

▶ **Definition 6** (MALL signature)**.** In order to interpret polarized MALL proofs, the following signature, and associated notations, are useful: $\mathcal{A} = \{\bot, \uparrow, \&_1, \&_2, \parr\}$ with $\mathrm{ar}(\bot) = 0, \mathrm{ar}(\uparrow) = \mathrm{ar}(\&_1) = \mathrm{ar}(\&_2) = 1, \mathrm{ar}(\parr) = 2$. When considering this signature, we write 1 rather than $\overline{\bot}$, $\downarrow$ rather than $\overline{\uparrow}$, $\oplus_i$ rather than $\overline{\&_i}$ and $\otimes$ rather than $\overline{\parr}$.

We define in the following the design η_F, the infinitary η-expansion of the axiom over F:

▶ **Definition 7.** Let F be a MALL formula. The design η_F is coinductively defined by:

$$\begin{array}{llll}
\eta_{F_1\otimes F_2} & = \eta_{F_1\parr F_2} & = & \parr(x_1, x_2).(x_0 \mid \otimes\langle \eta_{F_1,d}[x_1/x_0], \eta_{F_2}[x_2/x_0]\rangle) \\
\eta_{F_1\oplus F_2} & = \eta_{F_1\&F_2} & = & \sum_{i=1,2} \&_i(x_i).(x_0 \mid \oplus_i\langle \eta_{F_i}[x_i/x_0]\rangle) \\
\eta_{\downarrow F} & = \eta_{\uparrow F} & = & \uparrow(x_1).(x_0 \mid \downarrow\langle \eta_F[x_1/x_0]\rangle) \\
\eta_{\sigma Y.F} & = \eta_{F[\sigma Y.F/Y]} & \text{for } \sigma \in \{\mu, \nu\} &
\end{array}$$

▶ **Example 8.** Here are two additional examples of designs defined on the MALL signature:

$$d_1 = \&_1(x_1).(x_1 \mid 1) + \&_2(x_2).(x_2 \mid \oplus_1\langle\uparrow(y).(y \mid 1)\rangle)$$
$$d_2 = \parr(x_1, x_2).(x_2 \mid \downarrow\langle d_2\rangle)$$

▶ Remark. Designs (on the MALL signature) can be viewed as abstractions of (suitably polarized) MALL proofs. For instance d_1 abstracts the (unique) cut-free proof of $\vdash 1\&(\uparrow 1\oplus\bot)$.

The previous remark is the basis of the usual interpretation of MALL in ludics that we will extend, in the rest of the paper, into an interpretation of μMALLP. But first, as ludics is all about interaction, we turn to cut-elimination and orthogonality.

3.2 Cut-elimination and Orthogonality

Cuts can be reduced by the relation $\to$ defined as follows:

▶ **Definition 9.** The relation $\to$ is defined on positive designs as follows:

$$(\sum a(\vec{x}_a).p_a) \mid \overline{b}\langle\vec{n}\rangle \to p_b[\vec{n}/\vec{x}_b].$$

The reflexive and transitive closure of $\to$ is denoted $\to^\star$. We write $p \Downarrow q$ if $p \to^\star q$ and q is neither a cut, nor the design Ω. If such a design q does not exist, we write $p \Uparrow$. We define $\perp\!\!\!\perp$ to be the least set of positive designs containing $\maltese$ and closed by anti-reduction: $\perp\!\!\!\perp = \{d : d \to^\star \maltese\}$.

To eliminate cuts from designs, we coinductively propagate the relation $\Downarrow$ to subdesigns. The obtained normal form $(\!|d|\!)$ enjoys a weak form of Church-Rosser property.

▶ **Definition 10** (Normal form)**.** The function $(\!|.|\!)$ on designs is coinductively defined by:

$$\begin{array}{lll} (\!|p|\!) = \maltese & if & p \Downarrow \maltese; \\ \quad = \Omega & if & p \Uparrow; \\ \quad = x \mid \overline{a}\langle(\!|n_1|\!), \ldots, (\!|n_k|\!)\rangle & if & p \Downarrow x \mid \overline{a}\langle n_1, \ldots, n_k\rangle. \end{array} \qquad \begin{array}{l} (\!|x|\!) = x \\ (\!|\sum a(\vec{x}).p_a|\!) = \sum a(\vec{x}).(\!|p_a|\!). \end{array}$$

▶ **Proposition 11** (Associativity)**.** *Let d be a design and $n_1, \ldots, n_k$ be negative designs. One has:* $(\!|d[n_1/x_1, \ldots, n_k/x_k]|\!) = (\!|(\!|d|\!)[(\!|n_1|\!)/x_1, \ldots, (\!|n_k|\!)/x_k]|\!)$.

We finally define an orthogonality relation on so-called atomic designs.

▶ **Definition 12** (Atomic designs)**.** A positive standard design p is *atomic* if it has at most one free variable; that variable will be called x_0 in the rest of the paper. A negative standard design n is *atomic* if it is closed, *i.e.*, $\mathrm{fv}(n) = \emptyset$.

▶ **Definition 13.** Let p be a positive atomic design and n a negative atomic design. The designs p and n are said to be *orthogonal* (written $p \perp n$) if $p[n/x_0] \in \perp\!\!\!\perp$. Given a set $\mathbf{X}$ of atomic designs of the same polarity, we define its orthogonal $\mathbf{X}^\perp := \{\, e \mid \forall d \in \mathbf{X}, d \perp e \,\}$.

The orthogonality relation enjoys the usual properties:

▶ **Proposition 14.** *Let $\mathbf{X}$, $\mathbf{Y}$ be sets of atomic designs of the same polarity. One has:*

1) $\mathbf{X} \subseteq \mathbf{X}^{\perp\perp}$ 2) $\mathbf{X} \subseteq \mathbf{Y} \Rightarrow \mathbf{Y}^\perp \subseteq \mathbf{X}^\perp$ 3) $\mathbf{X}^\perp = \mathbf{X}^{\perp\perp\perp}$ 4) $(\mathbf{X} \cup \mathbf{Y})^\perp = \mathbf{X}^\perp \cap \mathbf{Y}^\perp$

3.3 Behaviours, Sets of Designs

Given a set $\mathbf{X}$ of atomic designs of the same polarity, $\mathbf{X}^\perp$ is the set of all those atomic designs that interact properly with respect to $\mathbf{X}$: they have a common behaviour with respect to the elements of $\mathbf{X}$. Proposition 14 ensures that such $\mathbf{X}^\perp$ are equal to their bi-orthogonal, this property characterizes them as *behaviours*:

▶ **Definition 15.** A *behaviour* is a set $\mathbf{X}$ of atomic designs of the same polarity such that $\mathbf{X} = \mathbf{X}^{\perp\perp}$. We denote by $\mathcal{C}_P$ (resp. $\mathcal{C}_N$) the set of all positive (resp. negative) behaviours.

$\mathcal{C}_P$, ordered by set inclusion, forms a complete lattice: using Proposition 14, we prove easily that every collection of positive behaviours $\vec{\mathbf{S}}$ has $(\bigcup \vec{\mathbf{S}})^{\perp\perp}$ as a least upper bound and $(\bigcap \vec{\mathbf{S}})^{\perp\perp} = \bigcap \vec{\mathbf{S}}$ as a greatest lower bound. Thus the Knaster-Tarski theorem guarantees the existence of least and greatest fixed points of monotonic operators on $\mathcal{C}_P$.

We generalize the relation $d \in \mathbf{C}$ between atomic designs and behaviours into the relation $d \models \mathbf{\Gamma}$ between designs with arbitrary free variables and contexts of behaviours:

▶ **Definition 16.** A *positive context* $\mathbf{\Gamma}$ is a set of pairs $x_1 : \mathbf{P_1}, \dots, x_k : \mathbf{P_k}$ where $x_1, \dots, x_k$ are distinct variables and $\mathbf{P_1}, \dots, \mathbf{P_k}$ are positive behaviours. A *negative context* $\mathbf{\Gamma}, \mathbf{N}$ is a positive context $\mathbf{\Gamma}$ together with a negative behaviour $\mathbf{N}$, to which no variable is associated.

▶ **Definition 17.** Let $\mathbf{\Gamma} = x_1 : \mathbf{P_1}, \dots, x_k : \mathbf{P_k}$ be a positive context, $\mathbf{\Gamma}, \mathbf{N}$ be a negative context, p (resp. n) be a positive (resp. negative) standard design. We define:

$$\begin{array}{lllll} p \models \mathbf{\Gamma} & \text{iff} & p[n_1/x_1, \dots, n_k/x_k] \in \perp\!\!\!\perp & \text{for any} & n_1 \in \mathbf{P_1}^\perp, \dots, n_k \in \mathbf{P_k}^\perp; \\ n \models \mathbf{\Gamma}, \mathbf{N} & \text{iff} & p[n[n_1/x_1, \dots, n_k/x_k]/x_0] \in \perp\!\!\!\perp & \text{for any} & p \in \mathbf{N}^\perp, n_1 \in \mathbf{P_1}^\perp, \dots, n_k \in \mathbf{P_k}^\perp. \end{array}$$

Remark that $p \models x_0 : \mathbf{P}$ if and only if $p \in \mathbf{P}$ and $n \models \mathbf{N}$ if and only if $n \in \mathbf{N}$. More generally, the following closure principle [12, 20] will be useful in the following sections.

▶ **Proposition 18** (Closure principle)**.**

$$\begin{array}{lll} d \models \mathbf{\Sigma}, z : \mathbf{P} & \textit{iff} & \forall m \in \mathbf{P}^\perp,\ (\!|d[m/z]|\!) \models \mathbf{\Sigma} \quad \textit{where } \mathbf{\Sigma} \textit{ is a positive or negative context.} \\ n \models \mathbf{\Gamma}, \mathbf{N} & \textit{iff} & \forall q \in \mathbf{N}^\perp,\ (\!|q[n/x_0]|\!) \models \mathbf{\Gamma} \quad \textit{where } \mathbf{\Gamma} \textit{ is a positive context.} \end{array}$$

3.4 Designs as Strategies

We end this background section on ludics by providing some more details on the comparison between HO game semantics [13] and ludics.

In HO game semantics, one first defines arenas which specify the moves of the game and which induce plays. In a second step, strategies are defined, as sets of plays satisfying various conditions (such as totality, determinism, innocence, *etc.*) depending on what is modeled. While arenas interpret formulas (or types), strategies will interpret proofs (or programs).

In ludics, the construction proceeds in the other direction, more akin to realizability models: a notion of abstract proof (design) serves as our notion of strategy while arenas are replaced by behaviours, that are sets of designs closed under bi-orthogonality. Strategies and interaction thus come first and only afterwards come the notion of arena: the moves of the game are defined as a by-product of the way the objects interact.

The comparison can be made more precise when comparing innocent game semantics and ludics [10, 5]. Indeed, with *innocent* strategies, a player's move does not depend on the full play that precedes it but only on a restriction of the play, its *view*. A view typically excludes the part of the play which corresponds to intermediate computations of the opponent, retaining only opponent's results and not how the results were obtained. As a consequence, innocent strategies can be presented as *sets of views* with some conditions. Ludics fits this presentation as designs can be seen as sets of views: each branch of a design is a view.

To conclude this comparison, let us stress that on the one hand, game semantics puts constraints on the way arenas are built but it is then rather flexible on the definition of strategies (by enforcing or relaxing various constraints on the structure of strategies). On the other hand, ludics puts constraints on the design of strategies (for instance to preserve analytical theorems on which internal completeness depends) and is quite flexible on how arenas are defined. This difference explains why it revealed to be much more difficult to model LL exponentials in ludics than in HO game semantics. The very same reason explains why it will be smoother to interpret fixed points in ludics than in HO game semantics [8]. We will come back to this last point when discussing related works in Section 6.

4 Interpretation of μMALLP in Ludics

We now define a semantics for our system in ludics, extending the usual interpretation of MALL in computational ludics [20]: formulas will be interpreted by behaviours and proofs by designs. From now on, we restrict to the MALL signature from Definition 6.

4.1 Interpretation of Formulas

▶ **Definition 19.** Let F be a preformula and $\mathcal{E}$ an *environment* mapping each free variable of F to a behaviour of the same polarity. We define by induction on F a behaviour called the *interpretation of F under $\mathcal{E}$* and denoted by $[\![F]\!]^{\mathcal{E}}$.

$[\![X]\!]^{\mathcal{E}} = \mathcal{E}(X) \quad [\![0]\!]^{\mathcal{E}} = \emptyset^{\perp\perp} \quad [\![1]\!]^{\mathcal{E}} = \{(x_0 \mid 1)\}^{\perp\perp} \quad [\![\downarrow N]\!]^{\mathcal{E}} = \{x_0 \mid \downarrow\langle r\rangle \,:\, r \in [\![N]\!]^{\mathcal{E}}\}^{\perp\perp}$
$[\![N_1 \otimes N_2]\!]^{\mathcal{E}} = \{(x_0 \mid \otimes\langle r_1, r_2\rangle) \,:\, r_i \in [\![N_i]\!]^{\mathcal{E}}\}^{\perp\perp}$
$[\![N_1 \oplus N_2]\!]^{\mathcal{E}} = \{(x_0 \mid \oplus_i\langle r_i\rangle) \,:\, i \in \{1,2\}, r_i \in [\![N_i]\!]^{\mathcal{E}}\}^{\perp\perp}$
$[\![\mu X.P]\!]^{\mathcal{E}} = \mathrm{lfp}(\Phi)$ where $\Phi : \mathcal{C}_P \longrightarrow \mathcal{C}_P,\ \mathbf{C} \longmapsto [\![P]\!]^{\mathcal{E},X\mapsto\mathbf{C}}$
$[\![N]\!]^{\mathcal{E}} = ([\![N^\perp]\!]^{\mathcal{E}})^\perp$ for all other cases

The interpretation of a formula F in the empty environment is simply written $[\![F]\!]$.

The well-definedness of the interpretation of $\mu X.P$ relies on the monotonicity of Φ, which is easily proved by induction on monotonic preformulas. Our interpretation of formulas enjoys the usual substitution property, which entails that the interpretation of fixed points is stable under unfolding.

▶ **Proposition 20.** *$[\![F[G/X]]\!]^{\mathcal{E}} = [\![F]\!]^{\mathcal{E},X\mapsto[\![G]\!]^{\mathcal{E}}}$ and $[\![\mu X.P]\!]^{\mathcal{E}} = [\![P[\mu X.P/X]]\!]^{\mathcal{E}}$.*

The interpretation of positive MALL formulas is made by biorthogonal closure and that of negative MALL formulas is made by orthogonal closure, so that the shape of the elements of the resulting sets is not obvious. Nevertheless, the internal completeness theorem of ludics [20] allows to characterize them. For example, if $p = x_0 \mid \overline{a}\langle\vec{n}\rangle \in [\![N_1 \otimes N_2]\!]$ then $\overline{a} = \otimes$, $\vec{n} = (n_1, n_2)$ and each $n_i \in [\![N_i]\!]$. In the same way, if $n = \sum a(\vec{x}_a).p_a \in [\![P_1 \parr P_2]\!]$ then $\vec{x}_\parr = (x_1, x_2)$ and $p_\parr \models x_1{:}[\![P_1]\!], x_2{:}[\![P_2]\!]$. The treatment of other MALL connectives can be found in [3], Proposition 51. Similarly, the interpretation of a ν formula is defined as the orthogonal of a least fixed point, but that is equivalent to the following more direct description as a greatest fixed point.

▶ **Proposition 21.** *$[\![\nu X.N]\!]^{\mathcal{E}} = \mathrm{gfp}(\Phi)$ where $\Phi : \mathcal{C}_N \to \mathcal{C}_N$ is such that $\Phi(\mathbf{C}) = [\![N]\!]^{\mathcal{E},X\mapsto\mathbf{C}}$.*

4.2 Interpretation of μMALLP Proofs

We interpret proofs compositionally, each rule corresponding to a construction on designs. Again, this extends the interpretation of MALL rules by Terui and Basaldella [6]. Proofs of positive sequents are going to be interpreted as positive designs, and similarly for negative sequents. In order to do so, we need to annotate positive formulas in sequents with distinct variable names. If $\Gamma = P_1, \ldots, P_n$ is a positive sequent, we say that $x_1 : P_1, \ldots, x_n : P_n$ is a *decoration* of Γ when the x_i are distinct positive variables. A decoration for a negative sequent Γ, N is obtained by adding N to a decoration of the positive part Γ.

We first give the final definition of the interpretation, in order to fix the ideas regarding its structure and purpose. Then we define the design construction $\mathbb{G}_{F,d}$ that is needed to interpret rule (ν).

▶ **Definition 22.** Let π be a proof of a sequent Γ, and Γ' a decoration of Γ. The interpretation of π in Γ' (written $[\![\pi]\!]^{\Gamma'}$) is defined by the rules of Figure 3. Each of these rules has the form

$$\frac{\{d_i \vdash \Gamma'_i\}_{i\in I}}{d \vdash \Gamma'}\ (r)$$

and stands for the following implication: If a proof π is obtained from the proofs $(\pi_i)_{i\in I}$ by applying rule (r), and $[\![\pi_i]\!]^{\Gamma'_i} = d_i$, then $[\![\pi]\!]^{\Gamma'} = d$.

Identity rules

$$\frac{}{\eta_P[x/x_0] \vdash x : P, P^\perp}\ (ax) \qquad \frac{n \vdash \Gamma, P^\perp \qquad d \vdash \Delta, x : P}{(\!|d[n/x]|\!) \vdash \Gamma, \Delta}\ (cut)$$

Fixed point rules

$$\frac{p \vdash \Gamma, x : P[\mu X.P/X]}{p \vdash \Gamma, x : \mu X.P}\ (\mu) \qquad \frac{n \vdash x : S, N[S^\perp/X] \qquad m \vdash \Gamma, S^\perp}{(\!|\mathbb{G}_{N,n}[m/x_0]|\!) \vdash \Gamma, \nu X.N}\ (\nu)$$

MALL rules

$$\frac{}{\Omega^- \vdash \Gamma, \top}\ (\top) \qquad \frac{p_1 \vdash \Gamma, x_1 : P_1 \qquad p_2 \vdash \Gamma, x_2 : P_2}{\&_1(x_1).p_1 + \&_2(x_2).p_2 \vdash \Gamma, P_1 \& P_2}\ (\&) \qquad \frac{n \vdash \Gamma, N}{x \mid \downarrow\langle n\rangle \vdash \Gamma, x : \downarrow N}\ (\downarrow)$$

$$\frac{p \vdash \Gamma}{\bot.p \vdash \Gamma, \bot}\ (\bot) \qquad \frac{p \vdash \Gamma, x_1 : P_1, x_2 : P_2}{\parr(x_1, x_2).p \vdash \Gamma, P_1 \parr P_2}\ (\parr) \qquad \frac{p \vdash \Gamma, x : P}{\uparrow(x).p \vdash \Gamma, \uparrow P}\ (\uparrow)$$

$$\frac{}{(x \mid 1) \vdash x : 1}\ (1) \qquad \frac{n_1 \vdash \Delta, N_1 \qquad n_2 \vdash \Gamma, N_2}{x \mid \otimes\langle n_1, n_2\rangle \vdash \Gamma, \Delta, x : N_1 \otimes N_2}\ (\otimes) \qquad \frac{n \vdash \Gamma, N_i}{x \mid \oplus_i\langle n\rangle \vdash \Gamma, x : N_1 \oplus N_2}\ (\oplus_i)$$

Figure 3 Interpretation of μMALLP proofs.

This interpretation may be understood by thinking of designs as proof terms: formulas are annotated by variables and proofs by designs in the same way that, in intuitionistic logic, hypotheses are annotated by variables and proofs by λ-terms.

The interpretation of rules (ax) and (cut) is quite natural. The axiom over P is interpreted by the copycat design η_P and cut is interpreted by the normal form of the cut between the interpretations of the two subproofs. The interpretation of MALL rules is the same as in [6]. The interpretation of the (μ) rule is trivial, based on the fact that fixed point unfolding is transparent in our interpretation. The main difficulty lies in the interpretation of the (ν) rule. Our goal is to interpret proofs by designs that reflect the computational behaviour of these proofs, thus we will derive the interpretation of rule (ν) from the cut reduction rule between μ and ν formulas presented in Figure 2. More precisely, our interpretation of rule (ν) is a design defined by an equality which expresses that the interpretation of the two proofs in Figure 2, which are obtained one from the other by cut elimination, are equal. As the reduction rule involves functoriality, our interpretation of rule (ν) is based on the construction $\mathbb{F}_{Q,d}$, the functoriality of Q applied to a design d, which is the counterpart in ludics of the construction $F_Q(\Pi)$, the functoriality of Q applied to a proof Π.

▶ **Definition 23.** Let d be a negative l-design and F a monotonic preformula such that $\mathrm{fv}(d) \subseteq \{x\}$ and $\mathrm{fv}(F) \subseteq \{X\}$. The *functoriality* of F applied to d is the negative l-design $\mathbb{F}_{F,d}$ coinductively defined by $\mathbb{F}_{F,d} = \eta_F$ when $fv(F) = \emptyset$, and otherwise:

$$\begin{array}{lcll}
\mathbb{F}_{X,d} & = & \mathbb{F}_{X^\perp,d} & = d[x_0/x] \\
\mathbb{F}_{F_1 \otimes F_2,d} & = & \mathbb{F}_{F_1 \parr F_2,d} & = \parr(x_1, x_2).(x_0 \mid \otimes\langle \mathbb{F}_{F_1,d}[x_1/x_0], \mathbb{F}_{F_2,d}[x_2/x_0]\rangle) \\
\mathbb{F}_{F_1 \oplus F_2,d} & = & \mathbb{F}_{F_1 \& F_2,d} & = \sum_{i=1,2} \&_i(x_i).(x_0 \mid \oplus_i\langle \mathbb{F}_{F_i,d}[x_i/x_0]\rangle) \\
\mathbb{F}_{\downarrow F,d} & = & \mathbb{F}_{\uparrow F,d} & = \uparrow(x_1).(x_0 \mid \downarrow\langle \mathbb{F}_{F,d}[x_1/x_0]\rangle) \\
\mathbb{F}_{\sigma Y.F,d} & = & \mathbb{F}_{F[\sigma Y.F/Y],d} & \text{for } \sigma \in \{\mu, \nu\}
\end{array}$$

The definition of functoriality in ludics naturally expresses the intended computational behaviour of that operation: $\mathbb{F}_{Q,d}$ is a modified η-expansion which behaves as d on occurrences

of X in Q. This should be contrasted with the very involved formulation of $F_Q(\Pi)$ in sequent calculus [2], which notably uses the (ν) rule to deal with fixed points encountered in Q.

▶ **Definition 24.** Let d be a negative design and F a monotonic preformula such that $\mathrm{fv}(d) \subseteq \{x\}$, $\mathrm{fv}(F) \subseteq \{X\}$ and $F \neq X$. The *action* of F on d is the design $\mathbb{G}_{F,d}$ coinductively defined by the following (productive) equation: $\mathbb{G}_{F,d} = \mathbb{F}_{F,\mathbb{G}_{F,d}}[d[x_0/x]/x_0]$.

4.3 Soundness and Invariance by Cut Elimination

Our first soundness result establishes that provability ($\vdash$) implies realizability ($\models$).

▶ **Definition 25.** If $\Gamma' = x_1 : P_1, \dots, x_n : P_n$ is a positive decorated sequent, we interpret it into the positive context $[\![\Gamma']\!] = x_1 : [\![P_1]\!], \dots, x_n : [\![P_n]\!]$. If Γ', N is a negative decorated sequent, its interpretation as a negative context is defined by $[\![\Gamma', N]\!] = [\![\Gamma']\!], [\![N]\!]$.

▶ **Theorem 26.** *If π is a proof of Γ, and Γ' is a decoration of Γ, then $[\![\pi]\!]^{\Gamma'} \models [\![\Gamma']\!]$.*

The theorem is proved by induction on π, and case analysis on its last rule. Soundness of rule (ax) follows from the fact that $\eta_P \models x_0 : [\![P]\!], [\![P]\!]^\perp$. Soundness for (cut) follows from the closure principle. The cases of MALL rules easily follow from the definition of formula interpretations. Soundness for rule (μ) is a direct consequence of Proposition 20. The difficulty lies in the (ν) rule, the soundness of which relies on the following proposition (proved in [3], Appendix A.1.1) stating that $\mathbb{F}_{F,d}$ is sound.

▶ **Proposition 27.** *Let d be a negative design, $\mathbf{P}, \mathbf{N}$ be two behaviours and F be a negative monotonic preformula such that $\mathrm{fv}(d) \subseteq \{x\}$, $\mathrm{fv}(F) \subseteq \{X\} \subseteq \mathcal{V}_N$ and $d \models x : \mathbf{P}, \mathbf{N}$. Then we have $(\!|\mathbb{F}_{F,d}|\!) \models x_0 : [\![F^\perp]\!]^{X \mapsto \mathbf{P}^\perp}, [\![F]\!]^{X \mapsto \mathbf{N}}$.*

▶ **Proposition 28.** *Let $\nu X.F$ be a formula and d a design such that $d \models x_0 : \mathbf{S}, [\![F]\!]^{X \mapsto \mathbf{S}^\perp}$. We have $(\!|\mathbb{G}_{F,d}|\!) \models x_0 : \mathbf{S}, [\![\nu X.F]\!]$.*

Proof. By closure principle, one has that $(\!|\mathbb{G}_{F,d}|\!) \models x_0 : \mathbf{S}, [\![\nu X.F]\!]$ iff $\forall m \in \mathbf{S}^\perp, (\!|\mathbb{G}_{F,d}[m/x_0]|\!) \models [\![\nu X.F]\!]$ iff $\mathbf{S}_1 = \{\ (\!|\mathbb{G}_{F,d}[m/x_0]|\!)\ :\ m \in \mathbf{S}^\perp\ \}^{\perp\perp} \subseteq [\![\nu X.F]\!]$. But $[\![\nu X.F]\!] = \mathrm{gfp}(\phi)$ where $\phi = \mathbf{C} \mapsto [\![F]\!]^{X \mapsto \mathbf{C}}$, thus it suffices to establish that $\mathbf{S}_1$ is a post-fixed point of ϕ, ie $\mathbf{S}_1 \subseteq [\![F]\!]^{X \mapsto \mathbf{S}_1}$. This is equivalent to $\forall m \in \mathbf{S}^\perp, (\!|\mathbb{G}_{F,d}|\!)[m/x_0] \models [\![F]\!]^{X \mapsto \mathbf{S}_1}$ and by closure principle to $(\!|\mathbb{G}_{F,d}|\!) \models x_0 : \mathbf{S}, [\![F]\!]^{X \mapsto \mathbf{S}_1}$. Remark that by definition of $\mathbf{S}_1$ we have $(\!|\mathbb{G}_{F,d}|\!) \models x_0 : \mathbf{S}, \mathbf{S}_1$. By Proposition 27, this gives us $(\!|\mathbb{F}_{F,\mathbb{G}_{F,d}}|\!) \models x_0 : [\![F^\perp]\!]^{X \mapsto \mathbf{S}^\perp}, [\![F]\!]^{X \mapsto \mathbf{S}_1}$. By hypothesis, $d \models x_0 : \mathbf{S}, [\![F]\!]^{X \mapsto \mathbf{S}^\perp}$ so by the closure principle we have, as expected:

$$(\!|\mathbb{G}_{F,d}|\!) = (\!|\mathbb{F}_{F,\mathbb{G}_{F,d}}[d/x_0]|\!) \models x_0 : \mathbf{S}, [\![F]\!]^{X \mapsto \mathbf{S}_1}$$ ◀

As a second soundness result, we show that our semantics is denotational, *i.e.*, the interpretation is invariant by cut elimination. The proof of this theorem relies on the following lemma (proved in [3], Appendix A.1.2) which expresses that ludics functoriality $\mathbb{F}_{F,d}$ is the semantical counterpart of the proof construction presented in Section 2.

▶ **Lemma 29.** *Let Π be a proof of $\vdash P, N$ and Q a negative monotonic preformula such that $\mathrm{fv}(Q) \subseteq \{X\} \subseteq \mathcal{V}_N$. One has $[\![F_Q(\Pi)]\!]^{x:Q^\perp[P^\perp/X], Q[N/X]} = (\!|\mathbb{F}_{Q,[\![\Pi]\!]^{x:P,N}}|\!)$.*

▶ **Theorem 30.** *If Π' is obtained from Π by μMALLP cut elimination rules, then $[\![\Pi]\!] = [\![\Pi']\!]$.*

5 On Completeness

Clearly, not all designs are interpretations of proofs, since some designs are not even recursive. More generally, it is highly non-trivial whether (or when) one can recover coinvariants from a design in order to finitely express it as a proof; indeed, coinvariants are completely hidden in the process of normalizing the interpretation of the (ν) rule. This is essentially the same difficulty that Girard encounters with second-order existential quantification in ludics [12], and which lead him to give a completeness result for Π_1 formulas only. In our setting, that would correspond to handle least fixed points only, which would be rather weak. Fortunately, we can do better thanks to our direct treatment of fixed points in the semantics.

We now introduce the class of essentially finite designs with respect to which we prove a completeness theorem in the rest of the paper.

▶ **Definition 31.** *Essentially finite designs* (EFD) are *inductively* defined by:

$$\begin{array}{rcl} p & ::= & (x \mid 1) \mid (x \mid \oplus_i \langle n \rangle) \mid (x \mid \otimes \langle n_1, n_2 \rangle) \mid (x \mid \downarrow \langle n \rangle) \\ n & ::= & \bot.p_0 \mid \&_1(x_1).p_1 + \&_2(x_2).p_2 \mid \parr(x_1, x_2).p \mid \uparrow(x_1).p_1 \mid \eta_F \mid \Omega^- \end{array}$$

with $x_1 \in fv(p_1)$, $x_2 \in fv(p_2)$ and $x_1, x_2 \in fv(p)$.

Essentially finite designs perform a finite computation (this is the MALL part) followed by a copycat. Even though they are inductively defined, EFDs can be infinite. In pure MALLP, proofs correspond exactly to EFDs. But, despite the fact that μMALLP extends MALLP, it is not obvious that completeness holds for EFDs in μMALLP. This is because the interpretation of μMALLP formulas yields more complex behaviours than with MALLP. As we shall see, we can still obtain this theorem, but it requires a bit of work.

▶ **Theorem 32.** *Let d be an essentially finite design, let Γ be a sequent and Γ' be a decoration of Γ. We have: $d \models \mathbf{\Gamma'}$ iff $d = [\![\pi]\!]^{\Gamma'}$ for some μMALLP proof π of Γ.*

The proof of this theorem is by induction on the structure of the EFD, using internal completeness. The only problematic case is when the EFD is an η-expansion: one needs to prove that if $\eta_F \models x_0 : Q, P^\perp$ then $\eta_F = [\![\pi]\!]^{x_0:Q,P^\perp}$ where π is a proof of $\vdash Q, P^\perp$. Observe that if $\eta_F \models x_0 : Q, P^\perp$ then F, P and Q have the same infinitary unfolding, hence $\eta_F = \eta_P = \eta_Q$. Moreover, it is easy to prove that $\eta_P \models x_0 : Q, P^\perp$ iff $[\![P]\!] \subseteq [\![Q]\!]$. Using these two observations, the η-expansion case amounts to proving the following theorem:

▶ **Theorem 33** (Completeness for semantic inclusion)**.** *Let P, Q be two positive formulas such that $[\![P]\!] \subseteq [\![Q]\!]$. There is a proof π of $\vdash Q, P^\perp$ satisfying $[\![\pi]\!]^{x_0:Q,P^\perp} = \eta_P$.*

The remainder of this paper is dedicated to the proof of this result. In order to study the provability of semantic inclusions in μMALLP, we shall introduce an intermediate, infinitary proof system S_∞. We prove that it is sound and complete for semantic inclusions (*i.e.*, $[\![F]\!] \subseteq [\![G]\!]$ iff $F \vdash G$ is derivable in S_∞) and that we can translate any S_∞ proof to a μMALLP derivation whose ludics interpretation is a copycat. We shall establish by the way that derivability in S_∞ and semantic inclusion are decidable.

5.1 The Infinitary Proof System S_∞

Our system S_∞ deals with two-sided sequents that always feature exactly one formula on each side. We first introduce S_∞ pre-proofs that are only locally sound, and then equip them with a validity condition that ensures soundness. This construction, as well as the resulting system, are very close to Santocanale's circular proofs [19].

$$\frac{}{F \vdash G}\ (A)$$

$$\frac{\{F_i \vdash G_i\}_{i\in[n]}}{s(F_1,\dots,F_n) \vdash\ s(G_1,\dots,G_n)}\ (s)$$

$$\frac{F[\sigma X.F/X] \vdash G}{\sigma X.F \vdash G}\ (\sigma_l)$$

$$\frac{F \vdash G[\sigma X.G/X]}{F \vdash \sigma X.G}\ (\sigma_r)$$

Figure 4 Infinitary proof system S_∞.

$$\dfrac{\dfrac{\dfrac{P(F\vdash G)}{F\vdash G}}{\uparrow F\vdash\uparrow G}\ (\uparrow) \qquad \dfrac{P(\uparrow F\vdash H)}{\uparrow F\vdash H}\ (\uparrow)}{\uparrow F\otimes\uparrow F\vdash\uparrow G\otimes H}\ (\otimes)$$

$$\dfrac{\dfrac{\dfrac{P(F\vdash G)}{F\vdash G}}{\uparrow F\vdash\uparrow G}\ (\uparrow) \qquad \dfrac{\dfrac{\uparrow F\vdash\uparrow(\uparrow G\otimes H)}{\uparrow F\vdash H}\ (\nu_r)}{\uparrow F\otimes\uparrow F\vdash\uparrow G\otimes H}\ (\otimes)}{\uparrow F\otimes\uparrow F\vdash G}\ (\mu_r)$$

$$\frac{\uparrow F\otimes\uparrow F\vdash G}{F\vdash G}\ (\mu_l)$$

$$\frac{\uparrow F\otimes\uparrow F\vdash\uparrow G\otimes H}{F\vdash\uparrow G\otimes H}\ (\mu_l)$$

Figure 5 Example of an S_∞ proof.

▶ **Definition 34.** S_∞ *pre-proofs* are trees coinductively generated from the rules of Figure 4. In that figure, $s(F_1,\dots,F_n)$ stands for a formula whose toplevel connective is a MALL connective s of arity n (*e.g.*, $\otimes$ has arity 2) and $\sigma \in \{\mu,\nu\}$. We say that a pre-proof is *fully justified* if it does not contain an application of rule (A).

The pre-proofs may be seen as η-expansions, but they are partial and unsound. Partiality comes from rule (A), which allows to make arbitrary assumptions. Unsoundness comes from the fact that, even without (A), pre-proofs are only locally sound. For example, $\mu X.{\downarrow}{\uparrow}X \vdash {\downarrow}\nu Y.{\uparrow}{\downarrow}Y$ admits a fully justified pre-proof.

▶ **Definition 35.** Let F, G be two formulas. $P(F \vdash G)$ is the S_∞ pre-proof of $F \vdash G$ coinductively defined by applying the first available rule in (σ_r), (σ_l), (s) or (A), and constructing the proofs of the premises $F_i \vdash G_i$ using the same construction $P(F_i \vdash G_i)$.

In other words, $P(F \vdash G)$ decomposes F and G as they agree on MALL connectives, giving priority to left unfolding of μ and ν. When MALL connectives become different, the pre-proof stops on an application of rule (A).

▶ **Example 36.** Let $F = \mu X.{\uparrow}X \otimes {\uparrow}X$, $G = \mu X.{\uparrow}X \otimes \nu Y.{\uparrow}({\uparrow}X \otimes Y)$ and $H = \nu Y.{\uparrow}({\uparrow}G \otimes Y)$. The pre-proof $P(F \vdash G)$ is given in Figure 5, where we have stopped expanding $P(F \vdash G)$ on sequents that have already occurred, explicitly showing the regular structure of the infinite proof. Note that this proof is fully justified.

We now turn to defining the validity condition that pre-proofs will have to satisfy in order to become proper proofs. Validity is based on parity conditions, as in Santocanale's work [19]. This requires a few preliminary definitions regarding subformulas. We denote by $\leq$ the subformula ordering, *i.e.*, $F \leq G$ if F is a subformula of G, and by $<$ the strict subformula ordering, *i.e.*, $F < G$ if $F \leq G$ and $F \neq G$.

▶ **Definition 37.** We define $\rightsquigarrow$ to be the least reflexive transitive relation on formulas such that: $s(F_1,\dots,F_n) \rightsquigarrow F_i$ and $\sigma X.F \rightsquigarrow F[(\sigma X.F)/X]$.

Note that, for a given F, there are only finitely many G such that $F \rightsquigarrow G$ (such formulas are in fact in bijection with the (open) subformulas of F). Also note that if $F \vdash G$ appears under $F' \vdash G'$ in a pre-proof, one has $F \rightsquigarrow F'$ and $G \rightsquigarrow G'$.

▶ **Proposition 38.** *For any cycle* $F_1 \rightsquigarrow F_2 \rightsquigarrow \dots F_n \rightsquigarrow F_1$ *there is some* $i \in [1;n]$ *such that* $F_i \leq F_j$ *for all* $j \in [1;n]$.

▶ **Definition 39** (Validity condition)**.** Let π be a pre-proof and γ an infinite branch of π. We define γ_r (resp. γ_l) to be the set of formulas appearing infinitely often on the right (resp. left) of sequents of γ. By Proposition 38, the elements of γ_r (resp. γ_l) have a minimum w.r.t. $\leq$; we note it $\min_r(\gamma)$ (resp. $\min_l(\gamma)$). It is easy to see that these minima are fixed point formulas. We say that the branch γ is *valid* if either $\min_l(\gamma)$ is a least fixed point or $\min_r(\gamma)$ is a greatest fixed point. We say that π is *valid* if all of its branches are valid.

▶ **Example 40.** The pre-proof in the previous example is valid, for the simple reason that on all branches, the formula F is unfolded infinitely often on the left of sequents. The pre-proof $P(\uparrow F \vdash H)$ would also be valid for the same reason, but $P(H \vdash \uparrow F)$ is not valid: the branch corresponding to taking the right of each tensor has only least fixed points on the right of its sequents and greatest fixed points on the left. Consider finally the pre-proof $P(G \vdash F)$. All of its branches that eventually always go to the right of tensors are invalid: on such branches, the minimum of formulas that occur infinitely often is H on the left of sequents and F on the right. All other branches, *i.e.*, those that go infinitely often to the left of tensors, are valid because G occurs infinitely often on the left of their sequents.

5.2 Completeness of S_∞

We first show that semantic inclusions are provable in S_∞: $[\![F]\!] \subseteq [\![G]\!]$ entails that $P(F \vdash G)$ is a valid, fully justified derivation. We prove that $P(F \vdash G)$ is fully justified by using internal completeness and Proposition 20. Proving its validity requires a few technical lemmas regarding the subformula ordering, that bridge the gap between the syntactic validity condition and the semantics of formulas.

▶ **Definition 41.** Let F, H be two preformulas, and X_0 a variable of the same polarity as H, not occurring in F nor H. We define $\mathcal{O}_H^{X_0}(F)$ as the unique preformula such that $\mathcal{O}_H^{X_0}(F)[H/X_0] = F$ and $H \not\leq \mathcal{O}_H^{X_0}(F)$. We shall simply write $\mathcal{O}_H(F)$ when the name of the variable is irrelevant or can be inferred from the context.

▶ **Proposition 42.** *Let F, H be two formulas such that $H < F$. For every MALL connective s and $\sigma \in \{\mu, \nu\}$, one has:*

- *If $F = s(F_1, \ldots, F_n)$ then $\mathcal{O}_H(s(F_1, \ldots, F_n)) = s(\mathcal{O}_H(F_1), \ldots, \mathcal{O}_H(F_n))$.*
- *If $F = \sigma Y.G$ then $\mathcal{O}_H(\sigma Y.G) = \sigma Y.\mathcal{O}_H(G)$ and unfolding F commutes with abstracting over H,* i.e.*, $\mathcal{O}_H(G[(\sigma Y.G)/Y]) = \mathcal{O}_H(G)[\mathcal{O}_H(\sigma Y.G)/Y]$.*

The proof of Proposition 42 can be found in [3], Appendix A.2.1.

▶ **Proposition 43.** *If $[\![F]\!] \subseteq [\![G]\!]$ then $P(F \vdash G)$ is a proof.*

Proof sketch. We prove the contrapositive. If $P(F \vdash G)$ is not fully justified, it is easy to show that $[\![F]\!] \not\subseteq [\![G]\!]$. Assume now that $P(F \vdash G)$ is fully justified but not valid. Then our derivation has an infinite branch $\gamma = (\gamma_k)_{0 \leq k} = (F_k \vdash G_k)_{0 \leq k}$ such that $F_l = \min_l(\gamma) = \nu X_l.K_l$ and $F_r = \min_r(\gamma) = \mu X_r.K_r$. Let d be the design that acts as an η-expansion along γ, and ✠ elsewhere, and let d_i be its subdesign corresponding to the subbranch of γ rooted in γ_i. Let I be the indices such that γ_i is the conclusion of an unfolding of F_l. We can show that, for all $i \in I$, $d_i \in [\![F_l]\!]$. This is proved by showing that $\mathbf{A} = \{\ d_i \mid i \in I\ \}^{\perp\perp}$ is a post-fixed point of $\phi : \mathbf{C} \mapsto [\![K_l]\!]^{X_l \mapsto \mathbf{C}}$, which amounts to proving that $\forall i \in I, d_i \in [\![K_l]\!]^{X_l \mapsto \mathbf{A}}$ or, equivalently, since $d_i = d_{i+1}$, that $\forall i \in I, d_{i+1} \in [\![\mathcal{O}_{F_l}(F_{i+1})]\!]^{X_l \mapsto \mathbf{A}}$. Generalizing this statement, we actually prove by induction that $\forall j \in \mathbb{N},\ d_j \in [\![\mathcal{O}_{F_l}(F_j)]\!]^{X_0 \mapsto \mathbf{A}}$, using Proposition 42. From there, we can show $d \in [\![F]\!]$ and, using a symmetry argument, $d \notin [\![G]\!]$, which concludes the proof. ◀

5.3 From S_∞ to μMALLP

We now prove that any valid fully-justified S_∞ proof can be transformed into a μMALLP proof. To prove this, we extend μMALLP with the rule (A) of Figure 4 and we call this system μMALLP*.

▶ **Definition 44.** Let π be a proof of a sequent s in S_∞ (resp. μMALLP*). We denote the set of sequents appearing in π as S_π, the conclusions of (A)-rules of π as $\mathcal{A}_\pi$ and call them the *assumptions* of π, and we let C_π be $S_\pi \setminus A_\pi$. The complexity of π is $\#\pi := card(C_\pi)$.

▶ **Definition 45.** Let F, G be two formulas and $\mathcal{H} \subseteq S_{P(F\vdash G)}$, $P(F \vdash G)^{\mathcal{H}}$ is the proof obtained from $P(F \vdash G)$ by replacing all the occurrences of the subtrees rooted in s by an assumption on s, for every $s \in \mathcal{H}$. (Notice that the subtrees rooted in s are all the same, and are equal to the tree $P(s)$.)

The result will follow from a slightly more general lemma:

▶ **Lemma 46.** *Let F, G be two formulas and $\mathcal{H} \subseteq S_{P(F\vdash G)}$. If $P(F \vdash G)^{\mathcal{H}}$ is valid then there is a proof π of $F \vdash G$ in $\mu MALLP^\star$ such that $\mathcal{A}_\pi \subseteq \mathcal{H}$.*

Proof sketch (see [3], Appendix A.2.2 for details). Let $s = F \vdash G$. The proof is by induction on $\#P(s)^{\mathcal{H}}$. Observe that if $\#P(s)^{\mathcal{H}} = 0$, then $s \in \mathcal{H}$ and the result obviously holds by using rule (A) on s in μMALLP*. In the inductive case, there are two possibilities:

1. There exist $s_1, s_2 \in C_{P(s)^{\mathcal{H}}}$ such that no occurrence of sequent s_1 appears above an occurrence of s_2 in $P(s)^{\mathcal{H}}$. In that case we decompose $P(s)^{\mathcal{H}}$ into $\Pi' = P(s)^{\mathcal{H}\cup\{s_2\}}$ and $\Pi'' = P(s_2)^{\mathcal{H}}$. Both Π' and Π'' have strictly smaller complexity than $P(s)^{\mathcal{H}}$. By induction hypothesis we obtain μMALLP* proofs π' of s and π'' of s_2, such that $\mathcal{A}_{\pi'} \subseteq \mathcal{H} \cup \{s_2\}$ and $\mathcal{A}_{\pi''} \subseteq \mathcal{H}$, which we plug together at the level of s_2 to get a μMALLP* proof of s.
2. Otherwise, we can find a valid branch containing all sequents appearing in $P(s)^{\mathcal{H}}$ (not as assumptions). This branch has either a least fixed point as minimum on the left of its sequents, or a greatest fixed point on the right. Assuming $\min_l(\gamma) = F_l = \mu X_l.K_l$ we decompose the proof at the unfoldings of F_l and design a suitable invariant in order to gather the pieces into a μMALLP* proof. ◀

When instanciating $\mathcal{H}$ to the empty set in Lemma 46 and remarking that a proof π in μMALLP* such that $\mathcal{A}_\pi = \emptyset$ is a μMALLP proof, we finally obtain:

▶ **Proposition 47.** *If $P(F \vdash G)$ is fully justified and valid then $F \vdash G$ is derivable in $\mu MALLP$.*

We can finally prove completeness for semantic inclusions.

Proof of Theorem 33. The result follows from Proposition 43 combined with a strengthening of Proposition 47 ensuring that $F \vdash G$ is provable by an η-expansion. To get this, notice that when we extend the syntax of designs, for every sequent s, by a negative constants A_s, and we interpret rule (A) by $\dfrac{}{A_s \vdash s}\ {}^{(A)}$, the interpretation of the μMALLP* proof of $F \vdash G$ constructed in lemma 46 is a partial η-expansion: indeed, we show that it is a copycat design which mimics $P(F \vdash G)^{\mathcal{H}}$ until reaching a sequent from $\mathcal{H}$ where it plays a constant A_s for some s in $\mathcal{H}$ (See [3], Proposition 52 for a detailed proof). As a corollary, when $\mathcal{H} = \emptyset$, this interpretation becomes a usual η-expansion, *i.e.*, without the constants A_s. ◀

5.4 Decidability of Semantic Inclusion

▶ **Proposition 48.** *If $P(F \vdash G)$ is fully justified and valid then $[\![F]\!] \subseteq [\![G]\!]$.*

Proof. We proceed essentially in the same way as in Lemma 46, proving the following generalization: Let F, G be two formulas and $\mathcal{H} \subseteq S_{P(F\vdash G)}$. If $P(F \vdash G)^{\mathcal{H}}$ is valid then, under the hypothesis that $\forall K \vdash L \in \mathcal{H}$, $[\![K]\!] \subseteq [\![L]\!]$, one has $[\![F]\!] \subseteq [\![G]\!]$. ◀

▶ **Proposition 49.** *Checking whether $P(F \vdash G)$ is valid is decidable.*

Decidability is proved by reducing the validity of all infinite branches to checking a finite number of combinations of elementary cycles, thanks to the fact that validity does not depend on the order in which elementary cycles are followed.

Proof. We extend the notations γ_r, γ_l to every finite path γ in $P(F \vdash G)$: γ_r (resp. γ_l) denotes the set of formulas appearing to the right (resp. left) of the sequents of γ. Then $\min(\gamma_r)$, $\min(\gamma_l)$ and the validity condition are the same as for infinite paths. Notice first that $P(F \vdash G)$ is not valid iff there exists a sequent s in $P(F \vdash G)$ and satisfying (P):

(P) There is an invalid finite path γ_1 in $P(s)$ from the root to an occurrence s.

Indeed, if there is such an s in $P(F \vdash G)$, and if γ_0 denotes a path in $P(F \vdash G)$ from the root to an occurrence of s, then the infinite path $\gamma_0\gamma_1^\omega$ is invalid. Conversely, let γ be an invalid infinite path in $P(F \vdash G)$, $F_l = \min(\gamma_l)$ and $G_r = \min(\gamma_r)$. There is a sequent s appearing infinitely often in γ such that the left-hand side of s is the formula F_l. As G_r appears infinitely often in γ_r, there is a finite sub-path of γ starting with s, ending with s and containing a sequent s' whose right-hand side is G_r. This finite path is obviously invalid, hence s satisfies (P).

We now prove that checking whether a sequent s satisfies (P) is decidable. Let $\delta_1, \ldots, \delta_n$ be the paths from the root of $P(s)$ to an occurrence of s which are of the form $\delta_i = s\sigma_i s$ and $s \notin \sigma_i$. We observe that every path γ from the root of $P(s)$ to an occurrence of s is a concatenation of some δ_i where $i \in I \subseteq [n]$, hence checking (P) amounts to find some $I \subseteq [n]$ such that $\min_{i\in I}(\min_l(\delta_i))$ is a ν formula and $\min_{i\in I}(\min_r(\delta_i))$ is a μ formula. This is obviously decidable.

Finally, since the number of sequents appearing in $P(F \vdash G)$ is finite and (P) is decidable, we conclude that validity is a decidable property for $P(F \vdash G)$. ◀

▶ **Theorem 50.** *Let F, G be two formulas. Checking whether $[\![F]\!] \subseteq [\![G]\!]$ is decidable.*

6 Conclusion

Contributions We have provided μMALLP with a denotational semantics in computational ludics. This construction is very natural, and did not require any change in the semantical framework to accommodate fixed points. Our interpretation gives an explicit formulation of the computational behaviour of μMALLP proofs as designs, which may provide a helpful alternative viewpoint to understand cut elimination in μMALLP. The fact that our model in ludics is relatively simple to work with has allowed us to venture into completeness investigations, a topic that is known to be tricky in presence of (co)induction. We have proved completeness for essentially finite designs using completeness of semantic inclusions for μMALLP . Technically, this last result uses, as an intermediate formalism, the infinitary system S_∞, which is very close to circular proofs with parity conditions. In order to prove completeness of μMALLP with respect to semantic inclusion, we proved completeness of μMALLP with respect to S_∞.

Related works. This last result is very much related to the work of Santocanale and Fortier [19, 11] who studied a circular proof system for a purely additive linear logic, equipped it with a cut elimination procedure, and gave a semantics of proofs in μ-bicomplete categories. Actually, the proof of Proposition 47 is inspired by Santocanale's argument [18] in his proof that circular proofs correspond to morphisms in μ-bicomplete categories. In fact, we could easily exploit his argument more generally, to translate to μMALLP a larger class of regular designs than just η-expansions.

Another obviously related work is Clairambault's game semantics for μLJ [8, 9], that is intuitionistic logic extended with least and greatest fixed points. In this semantics, he interprets (finite) proof objects as (infinite) winning strategies. More precisely, Clairambault first builds arenas with loops, simplifying McCusker's arenas for recursive types [16]. He then needs to equip the arenas with winning conditions for (finite and) infinite plays in such a way as to ensure that composition preserves totality. In ludics, the construction is simpler due to the fact that arenas, defined as behaviours, are rather secondary objects being generated from designs. Our construction is made particularly smooth by the fact that Terui's designs are general enough to interpret μMALL proofs and that the usual orthogonality (*i.e.*, interaction) of ludics is sufficient to forbid infinite chatterings that were causing the loss of totality in Clairambault's framework.

Facing the same difficulties as in our setting for getting completeness results for μLJ, Clairambault opts for a simpler approach in [9], proving a completeness result for $\mu\mathrm{LJ}^\omega$, an infinitary cut-free variant of μLJ. We can formulate and prove the same result in our framework. More generally, note that since Clairambault's game semantics for μLJ can be adapted to the linear case, it would be natural to compare precisely the interpretations of μMALL proofs in the two models.

Finally, our work is also related, though less closely, with the work of Brotherston and Simpson [7] who have recently explored the relationship between infinite, regular and finite proof systems for classical arithmetic, leaving open the question of the relative expressiveness of the regular and finite formalisms.

Future work. The most natural development of our work would be to extend Santocanale's work to the multiplicative case in order to obtain a circular presentation of full μMALLP. By doing so, we can hope to sharpen our completeness result on EFD by extended it to regular designs for which we conjecture a similar completeness theorem can be achieved. Another very interesting research direction would be to investigate under which conditions we can obtain a full abstraction result for our semantics.

Acknowledgments. We thank anonymous reviewers as well as Pierre Clairambault for their comments on this paper. The second author is funded by a grant from DIM RDM-IdF, Région Île–de-France. This work has been funded by ANR project ANR-14-CE25-0007 RAPIDO.

References

1 Andreas Abel, Brigitte Pientka, David Thibodeau, and Anton Setzer. Copatterns: programming infinite structures by observations. In *The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL'13, Rome*, pages 27–38. ACM, 2013.

2 David Baelde. Least and greatest fixed points in linear logic. *ACM Transactions on Computational Logic*, 13(1), April 2012.

3 David Baelde, Amina Doumane, and Alexis Saurin. Least and Greatest Fixed Points in Ludics. Technical Report hal-01178396, Inria, July 2015.

4 Gilles Barthe, Maria Joao Frade, Eduardo Giménez, Luis Pinto, and Tarmo Uustalu. Type-based termination of recursive definitions. *Mathematical Structures in Computer Science*, 14(01):97–141, 2004.

5 Michele Basaldella and Claudia Faggian. Ludics with repetitions (exponentials, interactive types and completeness). *Logical Methods in Computer Science*, 7(2), 2011.

6 Michele Basaldella and Kazushige Terui. On the meaning of logical completeness. In *Typed Lambda Calculi and Applications, 9th International Conference, TLCA 2009, Brasilia, Brazil*, pages 50–64, 2009.

7 James Brotherston and Alex Simpson. Sequent calculi for induction and infinite descent. *Journal of Logic and Computation*, 21(6):1177–1216, December 2011.

8 Pierre Clairambault. Least and greatest fixpoints in game semantics. In *12th International Conference on the Foundations of Software Science and Computational Structures (FOSSACS)*, volume 5504 of *Lecture Notes in Computer Science*, pages 16–31. Springer, March 2009.

9 Pierre Clairambault. Strong functors and interleaving fixpoints in game semantics. *RAIRO – Theor. Inf. and Applic.*, 47(1):25–68, 2013.

10 Claudia Faggian and Martin Hyland. Designs, disputes and strategies. In *Computer Science Logic, 16th International Workshop, CSL 2002, 11th Annual Conference of the EACSL, Edinburgh, Scotland, UK*, volume 2471 of *Lecture Notes in Computer Science*, pages 442–457. Springer, 2002.

11 Jérôme Fortier and Luigi Santocanale. Cuts for circular proofs: semantics and cut-elimination. In *Computer Science Logic 2013 (CSL 2013), Torino, Italy*, volume 23 of *LIPIcs*, pages 248–262. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2013.

12 Jean-Yves Girard. Locus solum. *Mathematical Structures in Computer Science*, 11:301–506, 2001.

13 Martin Hyland and Luke Ong. On full abstraction for PCF. *Information and Computation*, 163(2):285–408, December 2000.

14 Jean-Louis Krivine. Realizability in classical logic. *Panoramas et synthèses*, 27:197–229, 2009.

15 Ralph Matthes. Monotone (co)inductive types and positive fixed-point types. *ITA*, 33(4/5):309–328, 1999.

16 Guy McCusker. *Games and full abstraction for a functional metalanguage with recursive types.* CPHC/BCS distinguished dissertations. Springer, 1998.

17 N. P. Mendler. Inductive types and type constraints in the second order lambda calculus. *Annals of Pure and Applied Logic*, 51(1):159–172, 1991.

18 Luigi Santocanale. A calculus of circular proofs and its categorical semantics. Technical Report RS-01-15, BRICS, Department of Computer Science, University of Aarhus, May 2001.

19 Luigi Santocanale. A calculus of circular proofs and its categorical semantics. In *Foundations of Software Science and Computation Structures*, volume 2303 of *Lecture Notes in Computer Science*, pages 357–371. Springer, 2002.

20 Kazushige Terui. Computational ludics. *Theoretical Computer Science*, 412(20):2048–2071, 2011.

Modelling Coeffects in the Relational Semantics of Linear Logic*

Flavien Breuvart and Michele Pagani

Laboratoire PPS – Université Paris Diderot – Paris 7, France
{breuvart, pagani}@pps.univ-paris-diderot.fr

Abstract

Various typing system have been recently introduced giving a parametric version of the exponential modality of linear logic, e.g. [6, 2]. The parameters are taken from a semi-ring, and allow to express *coeffects* – i.e. specific requirements of a program with respect to the environment (availability of a resource, some prerequisite of the input, etc.).

We show that all these systems can be interpreted in the relational category (***Rel***) of sets and relations. This is possible because of the notion of multiplicity semi-ring, introduced in [3] and allowing a great variety of exponential comonads in ***Rel***. The interpretation of a particular typing system corresponds then to give a suitable notion of *stratification* of the exponential comonad associated with the semi-ring parametrising the exponential modality.

1998 ACM Subject Classification F.3.2 [Semantics of Programming Languages] Denotational semantics, F.4.1 [Mathematical Logic] Lambda calculus and related systems

Keywords and phrases relational semantics, bounded linear logic, lambda calculus

Digital Object Identifier 10.4230/LIPIcs.CSL.2015.567

1 Introduction

Various systems have been recently proposed based on a notion of parametrised exponential comonad [2, 6] in linear logic. The idea is to parametrise the of-course modality ! with elements taken from a semi-ring $\mathcal{S}$. The multiplicative monoid of $\mathcal{S}$ describes how the parameters interact under the comonad structure of ! (i.e. dereliction and digging) while the additive monoid of $\mathcal{S}$ gives the interaction under the monoidal structure of ! (i.e. weakening and contraction). The axioms of the semi-ring allow then to define a parametrised version of the usual rules of cut-elimination, preserving the confluence property (see Figure 2).

This approach is related with Girard, Scedrov and Scott's *bounded linear logic* (BLL) [8], and thus we refer to it as $\mathrm{B}_{\mathcal{S}}\mathrm{LL}$. It is in some sense both a generalisation and a restriction of BLL. It is a generalisation because it allows one to choose any semi-ring $\mathcal{S}$, as a parameter grammar, while BLL is given with respect to a fixed notion of parameters. On the other hand, $\mathrm{B}_{\mathcal{S}}\mathrm{LL}$ is a significant restriction because its parameters are just elements of the semi-ring $\mathcal{S}$ while BLL deals with first-order terms extending polynomials and allowing dependences.

The interest of $\mathrm{B}_{\mathcal{S}}\mathrm{LL}$ is to offer a logical ground to the design of type systems allowing to express various co-effects, that is requirements of a program with respect to the environment. For example, in [6] a semi-ring based on contractive affine transformations has been used to design a type system with annotations on the scheduling of processes; in [2], the semi-ring of non-negative real numbers is used to express the expected value of the number of times a probabilistic program calls its input during the evaluation. We briefly recall these examples

* Partially founded by French ANR project Coquas (number ANR 12 JS02 006 01).

24th EACSL Annual Conference on Computer Science Logic (CSL 2015).
Editor: Stephan Kreutzer; pp. 567–581

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

in Section 2.1. The interesting point is that although these type systems model quite different co-effects, their soundness is rooted in the same logical framework, that is $\mathrm{B}_{\mathcal{S}}\mathrm{LL}$.[1]

In this paper, we present various denotational models for $\mathrm{B}_{\mathcal{S}}\mathrm{LL}$. In the literature, there is a categorical axiomatisation of what is a model of $\mathrm{B}_{\mathcal{S}}\mathrm{LL}$ known by the name of *bounded exponential situation* (recalled here in Definition 5). This notion has been presented at first in [2], but it originates from Melliès' works on parametrised monads [10].[2] However there is no known concrete category satisfying the axioms of a bounded exponential situation: the paper [2] gives only a realisability model for few specific examples of semi-ring $\mathcal{S}$.

We give a general recipe for getting a bounded exponential situation out of a model of linear logic (Section 3 and Theorem 7). Intuitively, the main point of our construction is that $\mathrm{B}_{\mathcal{S}}\mathrm{LL}$ corresponds to a stratification of the exponential comonad along the semi-ring $\mathcal{S}$: any model of linear logic admitting such a stratification (and one model can admit more than one) defines a model also of $\mathrm{B}_{\mathcal{S}}\mathrm{LL}$. From our point of view, this result, although simple, can be seen as the first step in relating the semantical notion of "approximant" (or "stratus") of the linear logic exponential, with a notion of co-effect annotation in a type system.

In Section 4, we apply our recipe to the category ***Rel*** of sets and relations, showing various examples of bounded exponential situation. The category ***Rel*** provides one of the simplest models of linear logic, where the exponential comonad is given by the finite multiset functor. In fact, we consider a generalisation of this comonad given by the notion of *multiplicity semi-ring* in Carraro *et al.*'s [3]. A multiplicity semi-ring $\mathcal{R}$ is a semi-ring satisfying some properties (Definition 8) which generalise the notion of multiplicity given by the natural number semi-ring $\mathbb{N}$ in the finite multiset functor. This generalisation has been introduced in [3] for proving the existence of non-sensible models of the untyped λ-calculus in the category ***Rel***. As a by-product, the authors show how many and different can be the exponential comonads living in the category ***Rel***. We want to take advantage of this variety for giving relational models of $\mathrm{B}_{\mathcal{S}}\mathrm{LL}$, for any semi-ring $\mathcal{S}$.

We prove that one can stratify the exponential comonad associated with any multiplicity semi-ring $\mathcal{R}$ (so getting a model of $\mathrm{B}_{\mathcal{S}}\mathrm{LL}$) just by interpreting the parameter semi-ring $\mathcal{S}$ into the multiplicity semi-ring $\mathcal{R}$ (Theorem 11). Section 4.3 discusses some concrete examples of this construction, giving instances of $\mathcal{S}$ and $\mathcal{R}$.

Finally, Section 5 gives a taste of the fact that these constructions can be applied to different categories than ***Rel***. For example, we briefly discuss the case of models based on coherence spaces, giving stratifications of linear categories that are not compact closed.

2 Preliminaries

▶ **Definition 1.** A *semiring* is given by $(\mathcal{S}, \cdot, 1, +, 0)$ where $\mathcal{S}$ is a set, the sum $+$ is an associative commutative binary operation with a neutral element $0 \in \mathcal{S}$ and the product $\cdot$ is an associative binary operation distributing over $+$ (so 0 is absorbing for $\cdot$) and with a neutral element $1 \in \mathcal{S}$.

An *ordered semiring* $(\mathcal{S}, \cdot, 1, +, 0, \leq)$ is a semiring $(\mathcal{S}, \cdot, 1, +, 0)$ with a partial order $\leq$ such that sum and product are increasing monotone.

Notice that because of the monotonicity of the multiplication, $0 \leq 1$ (resp. $1 \leq 0$) implies

[1] Let us mention also [12, 13], giving several other examples of applications. These systems are not always described in the syntactical definition of $\mathrm{B}_{\mathcal{S}}\mathrm{LL}$, but they can be modeled in our concrete semantics.

[2] See also Melliès' presentation "Sharing and Duplication in Tensorial Logic" at the workshop *Developments in Implicit Computational complExity* 2013.

$$\frac{}{A \vdash A}\,\mathtt{Ax} \qquad \frac{\Gamma \vdash A \qquad \Delta, A \vdash B}{\Gamma, \Delta \vdash B}\,\mathtt{Cut}$$

$$\frac{\Gamma, A, B \vdash C}{\Gamma, A \otimes B \vdash C}\,\otimes\mathtt{L} \qquad \frac{\Gamma \vdash A \qquad \Delta \vdash B}{\Gamma, \Delta \vdash A \otimes B}\,\otimes\mathtt{R}$$

$$\frac{\Gamma \vdash A \qquad \Delta, B \vdash C}{\Gamma, \Delta, A \multimap B \vdash C}\,\multimap\mathtt{L} \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B}\,\multimap\mathtt{R}$$

$$\frac{\Gamma \vdash B}{\Gamma, A^0 \vdash B}\,\mathtt{Weak} \qquad \frac{\Gamma, A \vdash B}{\Gamma, A^1 \vdash B}\,\mathtt{Der} \qquad \frac{\Gamma, A^I, A^J \vdash B}{\Gamma, A^{I+J} \vdash B}\,\mathtt{Contr}$$

$$\frac{A_1{}^{I_1}, \cdots, A_n{}^{I_n} \vdash B}{A_1{}^{I_1 \cdot J}, \cdots A_n{}^{I_n \cdot J} \vdash B^J}\,\mathtt{J\text{–}Prom} \qquad \frac{\Gamma, A^I \vdash B \qquad J \geq I}{\Gamma, A^J \vdash B}\,\mathcal{S}\mathtt{wL}$$

Figure 1 The sequent calculus of $\mathrm{B}_{\mathcal{S}}\mathrm{LL}$. In a sequent $\Gamma \vdash A$, Γ is supposed to be a multiset of formulas (no implicit contraction rule is admitted).

that 0 is the bottom (resp. top) element of $\mathcal{S}$. However we will often consider examples of ordered semi-rings where the two neutral elements are incomparable. In [2] the authors impose 0 to be the bottom element, but this condition is not necessary.

▶ **Definition 2.** Given a set X and a semi-ring $\mathcal{S}$, we denote by $\mathcal{S}_f\langle X\rangle$ the set of functions $\mu : X \mapsto \mathcal{S}$ with finite support (where $\mathrm{supp}(\mu) = \{g \in X \mid \mu(g) \neq 0_{\mathcal{S}}\}$). We denote by $[\,]$ the constant function with value $0_{\mathcal{S}}$ and, for any $g \in X$, by $[g]$ the function with value $1_{\mathcal{S}}$ on g and $0_{\mathcal{S}}$ everywhere else.

▶ Remark. Any order on $\mathcal{S}$ implies an order on $\mathcal{S}_f\langle X\rangle$: $\mu \leq \nu$ iff $\forall g \in \mathrm{supp}(\mu), \mu(g) \leq_{\mathcal{S}} \nu(g)$.

▶ **Proposition 1.** *If $\mathbb{X}$ is a monoid then the set $\mathcal{S}_f\langle\mathbb{X}\rangle$ is endowed with a structure of semi-ring, defined by:*

$$0_{\mathcal{S}_f\langle\mathbb{X}\rangle} := [\,], \qquad (\mu +_{\mathcal{S}_f\langle\mathbb{M}\rangle} \nu)(g) := \mu(g) +_{\mathcal{S}} \nu(g),$$

$$1_{\mathcal{S}_f\langle\mathbb{X}\rangle} := [1_{\mathbb{X}}], \qquad (\mu \cdot_{\mathcal{S}_f\langle\mathbb{X}\rangle} \nu)(g) := \sum_{\substack{g', g'' \in \mathbb{X} \text{ s.t.} \\ g' \cdot_{\mathbb{X}} g'' = g}} \mu(g') \cdot_{\mathcal{S}} \nu(g''),$$

Notice that the sum appearing in the definition of $\mu \cdot_{\mathcal{S}_f\langle\mathbb{X}\rangle} \nu$ is well-defined because the supports of μ and ν are finite.

▶ **Definition 3.** Given an ordered semiring $\mathcal{S}$, we call $\mathrm{B}_{\mathcal{S}}\mathrm{LL}$ the logic given by:

- the *formulas* are defined by the grammar, with $J \in \mathcal{S}$:
  $A, B, C := \alpha \mid A \otimes B \mid A \multimap B \mid A^J$,
- the *sequent calculus* is given in Figure 1,
- the *cut-elimination procedure* is defined by the usual rules of multiplicative linear logic plus the rules of Figure 2.

One can add the additive connectives without any effort, we prefer however to omit their account because they do not play any crucial role with respect to our results. In [2], the authors use a term calculus instead of a logical sequent system: the two presentations can be made in relation via a Curry-Howard correspondence.

2.1 Examples

Trivial semi-ring: the multiplicative exponential fragment of intuitionistic linear logic is recovered from $\mathrm{B}_{\mathcal{S}}\mathrm{LL}$ by taking $\mathcal{S}$ as the one element semi-ring.

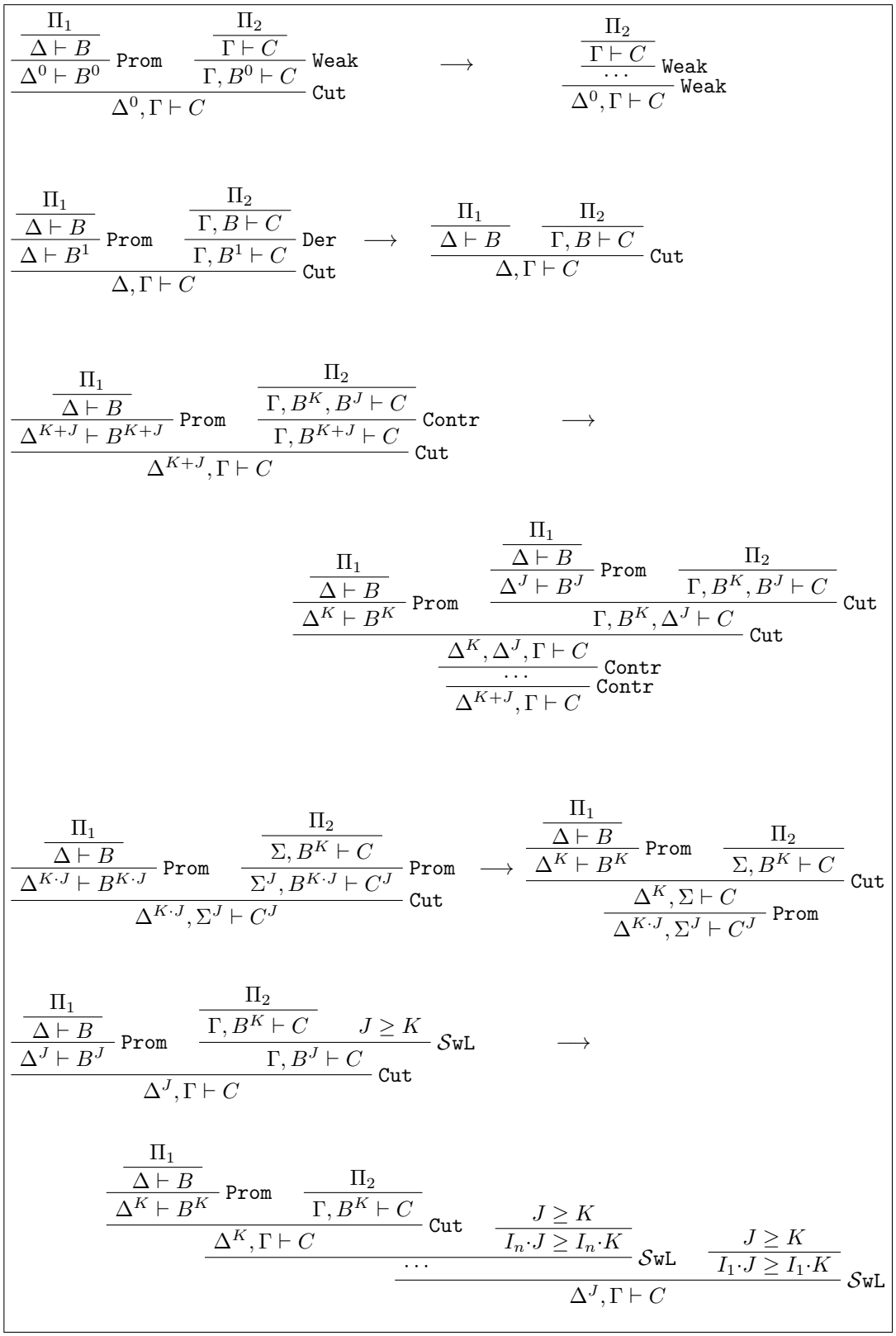


Figure 2 Cut-elimination rules (for the exponentials only). Given a sequent $\Delta = {A_1}^{I_1}, \ldots, {A_n}^{I_n}$ and a parameter J, we denote by Δ^J, the sequent ${A_1}^{J \cdot I_1}, \ldots, {A_n}^{J \cdot I_n}$. Notice in particular that $\Delta^0 = {A_1}^0, \ldots, {A_n}^0$, and $\Delta^1 = \Delta$.

Boolean semi-ring: the Boolean semi-ring $\mathbb{B} = (\{tt, ff\}, \wedge, tt, \vee, ff)$ allows finer types than the trivial one, distinguishing between data that can be weakened (of type A^{ff}) from data that can be duplicated (A^{tt}). The order over $\mathbb{B}$ plays a role, also: the discrete order will make the two types disjoint, while $tt \geq ff$ will make A^{ff} a subtype of A^{tt}, so that the $(\_)^{tt}$ modality behaves as the usual of-course modality ! of linear logic.

Natural numbers: the natural number semi-ring $(\mathbb{N}, \times, 1, +, 0)$ yields modalities expressing the number of times a resource is to be used. The order relation then allows some flexibility: for example, the natural order $0 < 1 < 2 < \dots$ makes A^n to be the type of data that can be used up-to n times. Notice that in this case there is no modality allowing a resource to be used an indefinite number of times, so the system is not an extension of linear logic. In order to recover the usual of-course modality ! one should take the order completion $\bar{\mathbb{N}}$, adding a top-element ω.

Polynomial semi-ring: by taking the semi-ring $(\mathbb{N}[X_i]_{i\in\mathbb{N}}, \times, 1, +, 0)$ of polynomials with natural numbers as coefficients (the choose order here is irrelevant for the discussion), one can express a basic form of resource dependency. One can write formulas like $A^{p(\vec{x})} \multimap B^{q(\vec{x})}$ where p, q are polynomials in the unknowns $\vec{x}$. Roughly speaking, this is the type of a function giving a result reusable $q(\vec{n})$ number of times as soon as its input can be used $p(\vec{n})$ number of times, for any sequence of natural numbers $\vec{n}$. This system has been discussed in [8] as an introduction to bounded linear logic (BLL). What is lacking with respect to the whole BLL is the possibility to bound first-order variables, so writing types of the form $A^{y\leq p(x)}$, where y is an unknown of a polynomial occurring inside A.

Affine contractive transformations: the one-dimensional contractive affine transformations $x \mapsto sx + p$ can be represented by real-valued matrices $x_{s,p} = \left(\begin{smallmatrix} s & p \\ 0 & 1 \end{smallmatrix}\right)$ with $0 \leq s \leq 1$ and $-1 \leq s + p \leq 1$. The value s is a scaling factor relative to the unit interval, and p is a delay from the time origin. The set of such transformations forms a monoid Aff_1^c with composition given by matrix product.[3] By Proposition 1, $\mathbb{N}_f\langle \mathrm{Aff}_1^c \rangle$ is a semi-ring so it defines the logic $\mathrm{B}_{\mathbb{N}_f\langle \mathrm{Aff}^c{}_1\rangle}\mathrm{LL}$. This system has been introduced by Ghica and Smith [6] in order to express at the level of types a scheduling on the execution of certain resources. For example, a formula $A^{\left[\left(\begin{smallmatrix} .5 & 0 \\ 0 & 1 \end{smallmatrix}\right), \left(\begin{smallmatrix} .5 & .25 \\ 0 & 1 \end{smallmatrix}\right)\right]}$ represents a resource of type A that can be called twice, both calls will last $\frac{1}{2}$ the duration relative to which we are measuring, but one call starts at the beginning of the available time interval, while the other call starts when $\frac{1}{4}$ of the time has elapsed. Of course, such annotations have a meaning when the language has primitives describing processes to be scheduled. See [6] for more details.

Positive real numbers: in presence of random primitives, one can associate any resource with a discrete random variable quantifying on the number of times this resource is used during the evaluation. In [2], $\mathrm{B}_{\mathcal{S}}\mathrm{LL}$ has been parametrised with the ordered semi-ring $\mathbb{R}^+ = (\mathbb{R}^+, \times, 1, +, 0, \leq)$ of the non-negative real numbers endowed with the natural order, the parameters expressing the expected values of these random variables.

This system can be extended (syntactically) with true dependent types and be able to catch finer properties, like differential privacy [5].

[3] Notice, however, that the semring product $I \cdot J$ denotes the reverse matrice product $J \cdot I$, this is due to a change in notation between us and [6].

3 Stratifying Linear Logic Exponentials

We recall the notion of *linear category*, which has been introduced in [1] as a categorical axiomatization of a model of intuitionistic linear logic. This definition has been recently revisited in [2] with the notion of *bounded exponential situation*, which roughly corresponds to a variant of linear category where the exponentials are parametrised by the elements of a partially ordered semi-ring $\mathcal{S}$ and which gives a categorical model of $\mathrm{B}_{\mathcal{S}}\mathrm{LL}$. Our contribution is the definition of *stratification* (Definition 6), giving a general recipe for extracting a bounded exponential situation from a linear category (Theorem 7). Section 4 will apply this recipe to the concrete case of the relational category.

▶ **Definition 4** ([1])**.** A *linear category* consists of:

- a symmetric monoidal closed category $(\mathcal{L}, \otimes, \mathtt{1}, \multimap)$,
- a comonad $(!, \mathtt{d} : !A \to A, \mathtt{p} : !A \to !!A)$ endowed with three natural transformations and a morphism: $\mathtt{w}_A : !A \to \mathtt{1}$, $\mathtt{c}_A : !A \to !A \otimes !A$, $\mathtt{m}_{\mathtt{1}} : \mathtt{1} \to !\mathtt{1}$, $\mathtt{m}_{A,B} : !A \otimes !B \to !(A \otimes B)$, satisfying a bunch of equations (see, for example, [1]).

A linear category $\mathcal{L}$ gives a model of $\mathrm{B}_{\mathcal{S}}\mathrm{LL}$ with $\mathcal{S}$ the trivial semi-ring (i.e. the usual intuitionistic MELL). In this case we have just one exponential modality, which is interpreted by the functor ! of $\mathcal{L}$ and its associated structure. When $\mathcal{S}$ is non-trivial, one has to parametrise the exponential modality ! by the elements of $\mathcal{S}$ and to add some equations making to interact these various modalities following the laws of the semi-ring $\mathcal{S}$. Such a structure has been suggested by Melliès and formally introduced in [2]:

▶ **Definition 5** ([2])**.** A *bounded exponential situation* consists of:

- a symmetric monoidal closed category $(\mathcal{L}, \otimes, \mathtt{1}, \multimap)$, used to interpret the multiplicative fragment of $\mathrm{B}_{\mathcal{S}}\mathrm{LL}$;
- a categorical axiomatization of the notion of partially ordered semi-ring, that is a bimonoidal category $(\mathcal{S}, +, 0, \cdot, 1)$. The objects $I, J, H, \ldots$ of $\mathcal{S}$ correspond to the elements of the semi-ring and the hom-sets define the order of the semiring: $I \leq J$ iff $\mathcal{S}(I, J)$ is non-empty[4].
- an exponential action $(\_)^{-}$ of $\mathcal{S}$ on $\mathcal{L}$, giving a parametric version of the exponential comonad and used to interpret the formulas A^I. Formally, it is a bifunctor $(\_)^{-} : \mathcal{S} \times \mathcal{L} \to \mathcal{L}$ together with six natural transformations: $\mathtt{p}'_{I,J,A} : A^{I \cdot J} \to (A^J)^I$, $\mathtt{d}'_A : A^1 \to A$, $\mathtt{w}'_A : A^0 \to \mathtt{1}$, $\mathtt{c}'_{I,J,A} : A^{I+J} \to A^I \otimes A^J$, $\mathtt{m}'_{I,\mathtt{1}} : \mathtt{1} \to \mathtt{1}^I$, $\mathtt{m}'_{I,A,B} : A^I \otimes B^I \to (A \otimes B)^I$, which should satisfy various diagrams, see [2] for details.

▶ **Definition 6.** A *stratification of a linear category* $\mathcal{L}$ is a triplet $(\mathcal{S}, (\_)^{-}, \varrho)$, where:

- $\mathcal{S}$ is an ordered semi-ring (seen as a bimonoidal category);
- $(\_)^{-}$ is a bifunctor $\mathcal{S} \times \mathcal{L} \to \mathcal{L}$;
- ϱ is a natural transformation from ! to $(\_)^{-}$, i.e. $\varrho_{I,A} : !A \mapsto A^I$ such that:
  - each of the morphism $\varrho_{I,A} : !A \mapsto A^I$ is an epimorphism, *i.e.*, for any $\phi, \psi : A^I \to B$, if $\varrho_{I,A}; \phi = \varrho_{I,A}; \psi$ then $\phi = \psi$,
  - the diagrams of Figure 3 can be completed (in a unique way due to the epi property) by families of morphisms $\mathtt{d}'_A, \mathtt{p}'_{A,I,J}$, $\mathtt{w}'_A$, $\mathtt{c}'_{A,I,J}$ and $\mathtt{m}'_{A,B,I,J}$, for any A, B, I, J.

[4] For our purposes, we can suppose that $\mathcal{S}$ has hom-sets of at most one element.

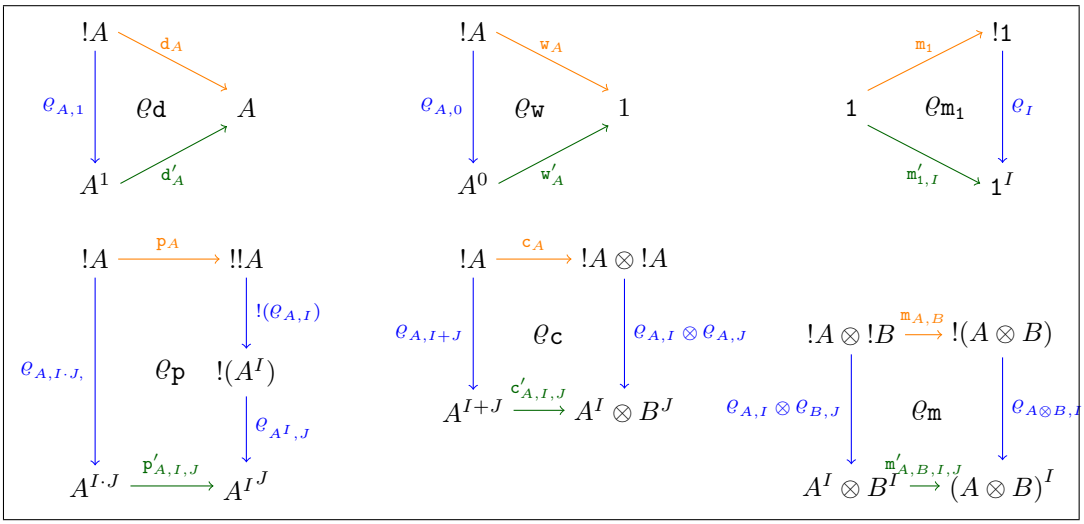


Figure 3 Coherence diagrams between the natural transformation ϱ, the exponential structure $(!, \mathtt{d}, \mathtt{p}, \mathtt{w}, \mathtt{c}, \mathtt{m})$ of a linear category and the exponential structure $((\_)^-, \mathtt{d}', \mathtt{p}', \mathtt{w}', \mathtt{c}', \mathtt{m}')$ of a bounded exponential situation.

Notice that all the diagrams of Figure 3 simply state that each natural transformation e required for the linearity of $\mathcal{L}$ is transported along ϱ to its parametrized version e'. Notice also that the diagram $\varrho\mathtt{m}_1$ of Figure 3 is always obtained for $\mathtt{m}'_I := \mathtt{m}_1; \varrho_{1,I}$.

Finally, the naturality of the families $\mathtt{d}'_A, \mathtt{p}'_{A,I,J}, \mathtt{w}'_A, \mathtt{c}'_{A,I,J}$ and $\mathtt{m}'_{A,B,I,J}$ can be automatically retrieved from the diagrams of Figure 3 and the universal property of epimorphisms.

▶ **Theorem 7.** *A stratification $(\mathcal{S}, (\_)^-, \varrho)$ of a linear category yields a bounded exponential situation hence a model of* $\mathrm{B}_{\mathcal{S}}\mathrm{LL}$.

Proof. The transformations defining a bounded exponential situation are given by the families $\mathtt{d}'_A, \mathtt{p}'_{A,I,J}, \mathtt{w}'_A, \mathtt{c}'_{A,I,J}$ and $\mathtt{m}'_{A,B,I,J}$. In fact, the naturality and coherence diagrams associated with these transformations are obtained by invoking the corresponding diagram from the linear category, by transporting the whole diagram through ϱ (via pre/post composing and the diagrams of Figure 3), and finally by using the universal property of epimorphisms.

For example, Figure 4 gives the commutation that the morphism $\mathtt{p}'_{A,I,J}$ should enjoy in order to give a positive action. The triangle **I** is naturality of ϱ over the associativity of the semiring multiplication, the square **IV** is the usual one of a linear category, **V** uses the promotion of the square on the first line of Figure 3, **VI** and **VII** are the naturality of, resp., ϱ and $\mathtt{p}$, and finally **II**, **III** and **VIII** are again squares of Figure 3. Notice that this is *a priori* not sufficient to obtain the commutation of the external cell due to the first ϱ that point on the wrong direction. However, we actually obtain that $\varrho; A^{\mathtt{as}^{\cdot}}; \mathtt{p}'; \mathtt{p}'^{H} = \varrho; \mathtt{p}'; \mathtt{p}'$ which results in the commutation of the external cell by the universal property of the epi ϱ. ◀

4 Relational Based Models

4.1 The linear category $\boldsymbol{Rel}^{\mathcal{R}}$

The category ***Rel*** has sets as objects and relations as morphisms, i.e. $\boldsymbol{Rel}(X, Y) := \mathcal{P}(X \times Y)$. Composition and identities are given by:

$$f; g := \{(x, y) \mid \exists z, (x, z) \in f, (z, y) \in g\}, \qquad id_X := \{(x, x) \mid x \in X\}.$$

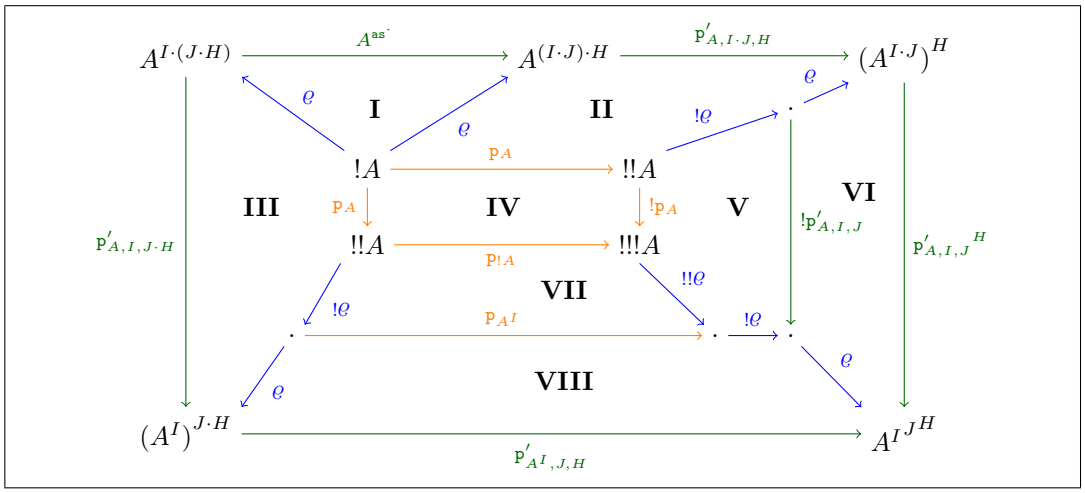


Figure 4 An example of the proof of the commutation of the diagrams needed to have a bounded exponential situation.

Rel is symmetric monoidal closed (in fact, compact closed) with tensor product given by:

$$X \otimes Y := X \times Y, \qquad f \otimes g := \{((x,x'),(y,y')) \mid (x,y) \in f,\ (x',y') \in g\}.$$

Associativity ($\alpha^{\otimes}_{X,Y,Z} := \{((x,(y,z)),((x,y),z)) \mid x \in X, y \in Y, z \in Z\} \in \boldsymbol{Rel}(X \otimes (Y \otimes Z),(X \otimes Y) \otimes Z)$) and commutativity ($\beta^{\otimes}_{X,Y} := \{((x,y),(y,x)) \mid x \in X, y \in Y\} \in \boldsymbol{Rel}(X \otimes Y, Y \otimes X)$) are natural bijections; the neutral object is the singleton $1 := \{*\}$. The internal hom functor is defined by: $X \multimap Y := X \otimes Y$ and $f \multimap g := f \otimes g$, the evaluation morphism is $eval := \{(((x,y),x),y) \mid x \in X, y \in Y\} \in \boldsymbol{Rel}((X \multimap Y) \otimes X, Y)$.

It is well-known that ***Rel*** models the linear logic exponential with the multi-set comonad. It is less known, however, that this is just an example of how one can express the exponential modality. Carraro et al. [3] have shown many other possibilities by introducing the notion of resource semi-ring (here Definition 8 and Theorem 9). We briefly recall this result, adding some examples. [5]

▶ **Definition 8** ([3]). A *multiplicity semi-ring* is a semi-ring $\mathcal{R} = (|\mathcal{R}|, \cdot, 1, +, 0)$ such that (p, q, r will vary over $\mathcal{R}$):

(MS1) $\mathcal{R}$ is *positive*: $p+q = 0$ implies $p = q = 0$;

(MS2) $\mathcal{R}$ is *discrete*: $p+q = 1$ implies $p = 0$ or $q = 0$;

(MS3) $\mathcal{R}$ is *additively splitting*: $p_1 + p_2 = q_1 + q_2$ implies $\exists r_{1,1}, r_{1,2}, r_{2,1}, r_{2,2}$, such that

$$p_i = r_{i,1} + r_{i,2}\ , \qquad q_i = r_{1,i} + r_{2,i}\ ;$$

(MS4') $\mathcal{R}$ is *multiplicatively splitting*: $q_1 + q_2 = p \cdot r$ implies there is $l \in \mathbb{N}$ such that for all $j \leq l$, we can find $r_j, p_{1,j}, p_{2,j}$ such that

$$\begin{aligned} r &= r_1 + \cdots + r_l, \\ p &= p_{1,j} + p_{2,j} \qquad \text{for all } j \leq l, \\ q_i &= p_{i,1} \cdot r_1 + \cdots + p_{i,l} \cdot r_k. \end{aligned}$$

[5] Notice once again that we have to reverse the semiring multiplication of [3] due to a change in notations.

The notion of multiplicity semi-ring given by Definition 8 is a slight generalization of the one in [3], because the multiplicative splitting has been slightly relaxed. It is straightforward to check that all proofs in [3] still hold.

The semi-ring of natural numbers $\mathbb{N}$ is the prototypical example of multiplicity semi-ring, while the Boolean semi-ring (as well as any cyclic semi-ring) is a non-example because the discreteness condition fails. Other non-trivial examples can be obtained via the following propositions.

▶ **Proposition 2.** *For any multiplicity semi-ring $\mathcal{R}$, the extension with an idempotent (for $+$ and $\cdot$) element ω that is absorbing for the addition and the multiplication (except with 0) results in a semiring $\bar{\mathcal{R}} = \mathcal{R} \cup \{\omega\}$ that is a multiplicity semi-ring.*

For example, the semi-ring $\bar{\mathbb{N}} = \mathbb{N} \cup \{\omega\}$ is a multiplicity semi-ring. The idea is that it allows an infinite number of resources (and by infinite we do not mean unbounded, but really infinite, obtained by taking a greatest fixpoint for example).

▶ **Proposition 3.** *Given a monoid $\mathbb{M}$, the semi-rings $\mathbb{N}_f\langle\mathbb{M}\rangle$ and $\bar{\mathbb{N}}_f\langle\mathbb{M}\rangle$ are multiplicity semi-rings.*

For example, the semi-ring $\mathbb{N}_f\langle\mathsf{Aff}_1^c\rangle$ induced by the monoid Aff_1^c of one-dimensional affine contractive transformations [6] is an example of multiplicity semi-ring, different from $\mathbb{N}$.

▶ **Theorem 9** (***Rel***$^{\mathcal{R}}$, [3])**.** *Any multiplicity semi-ring $\mathcal{R}$ defines an exponential comonad over* ***Rel*** *(for $r \in \boldsymbol{Rel}(A,B)$):*

$$!_{\mathcal{R}}A := \mathcal{R}_f\langle A\rangle,$$
$$!_{\mathcal{R}}r := \{(u,v) \in \boldsymbol{Rel}(!_{\mathcal{R}}A, !_{\mathcal{R}}B) \mid \exists \sigma \in \mathcal{R}_f\langle r\rangle,\ u(a) = \sum_{b\in B} \sigma(a,b),\ v(b) = \sum_{a\in A} \sigma(a,b)\}$$

Dereliction $\mathtt{d}_A := \{(\delta_a, a) \mid a \in A\} : !_{\mathcal{R}}A \to A$, where $\delta_a(a) = 1$ and $\delta_a(a') = 0$ for every $a \neq a'$, digging $\mathtt{p}_A := \{(m, M) \mid \forall a \in A, m(a) = \Sigma_{n\in!_{\mathcal{R}}A} n(a)\cdot M(n)\} : !_{\mathcal{R}}A \to !_{\mathcal{R}}!_{\mathcal{R}}A$, contraction $\mathtt{c}_A := \{(u,(v_1,v_2)) \mid \forall a \in A, u(a) = v_1(a) + v_2(a)\} : !_{\mathcal{R}}A \to !_{\mathcal{R}}A \otimes !_{\mathcal{R}}A$, weakening $\mathtt{w}_A = \{(0,*)\} : !_{\mathcal{R}}A \to \mathtt{1}$, where 0 denotes the constant zero function in $\mathcal{R}_f\langle A\rangle$, and the morphisms $\mathtt{m}_1 = \{(*,u) \mid u \in !_{\mathcal{R}}\mathtt{1}\} : \mathtt{1} \to !_{\mathcal{R}}\mathtt{1}$ and $\mathtt{m}_{A,B} := \{((u_1,u_2),v) \mid u_1(a) = \Sigma_b v(a,b),\ u_2(b) = \Sigma_a v(a,b)\} : (!_{\mathcal{R}}A \otimes !_{\mathcal{R}}B) \to !_{\mathcal{R}}(A \otimes B)$ are natural and respect usual diagrams.

We denote by ***Rel***$^{\mathcal{R}}$ *the linear category induced by this exponential comonad.*

A construction due to Grellois and Melliès [9] can be used to extend these results in any semi-ring $\mathcal{R}$ of the form $\mathcal{R}'_f\langle\mathbb{M}\rangle$ (where $\mathcal{R}'$ is a multiplicity semi-ring and $\mathbb{M}$ a monoid). This uses the fact that $\mathcal{R}'_f\langle\mathbb{M}\rangle$ is a composition of the exponential comonad $!_{\mathcal{R}}$ and a writer comonad $(\mathbb{M}, \cdot, \otimes)$ that distributes over the former.

4.2 Stratifications over ***Rel***$^{\mathcal{R}}$

We show how to associate with an ordered semi-ring $\mathcal{S}$ a stratification of the linear category ***Rel***$^{\mathcal{R}}$, for any multiplicity semi-ring $\mathcal{R}$. The key-point is that such stratifications can be presented as a kind of interpretation of the ordered semi-ring $\mathcal{S}$ into the hom-set $\boldsymbol{Rel}^{\mathcal{R}}(!_{\mathcal{R}}\mathtt{1}, \mathtt{1})$, which is isomorphic to the power-set $\mathcal{P}(\mathcal{R})$. Definition 10 gives the conditions that such interpretation should enjoy in order to induce a stratification over ***Rel***$^{\mathcal{R}}$ (Theorem 11).

Given a semi-ring $\mathcal{R}$, one can define the following operations over $\mathcal{P}(\mathcal{R})$ (α, β, γ vary over $\mathcal{P}(\mathcal{R})$):

$$\alpha \oplus \beta := \{p + q \mid p \in \alpha, q \in \beta\},$$
$$\alpha \odot \beta := \left\{\sum_{i=1}^{h} p_i \cdot q_i \mid h \geq 0, \sum_{i=1}^{h} q_i \in \beta, \forall i \leq h, p_i \in \alpha\right\}.$$

The operation $\oplus$ (resp. $\odot$) will be used to stratify contraction (resp. digging). Notice that the two operations are associative, $\oplus$ is commutative but not $\odot$, $\{0_{\mathcal{R}}\}$ (resp. $\{1_{\mathcal{R}}\}$) is the neutral element of $\oplus$ (resp. $\odot$). Moreover, $\odot$ left-distributes over $\oplus$ (i.e. $\gamma \odot (\alpha \oplus \beta) = (\gamma \odot \alpha) \oplus (\gamma \odot \beta)$), but it does not right-distribute. For example, take $\mathcal{R}$ to be the standard semi-ring over natural numbers, then:

$$(\{1\} \oplus \{1\}) \odot \{1, 2\} = \{2, 4\}, \qquad (\{1\} \odot \{1, 2\}) \oplus (\{1\} \odot \{1, 2\}) = \{2, 3, 4\}.$$

▶ **Definition 10.** An *interpretation* of an ordered semi-ring $\mathcal{S}$ into a multiplicity semi-ring $\mathcal{R}$ is a function $[\![-]\!] : \mathcal{S} \mapsto \mathcal{P}(\mathcal{R})$ such that (for all $I, J \in \mathcal{S}$):

$$I \leq_{\mathcal{S}} J \text{ implies } [\![I]\!] \subseteq [\![J]\!], \qquad [\![I]\!] \oplus [\![J]\!] \subseteq [\![I +_{\mathcal{S}} J]\!], \qquad [\![I]\!] \odot [\![J]\!] \subseteq [\![I \cdot_{\mathcal{S}} J]\!],$$
$$\{0_{\mathcal{R}}\} \subseteq [\![0_{\mathcal{S}}]\!], \qquad \{1_{\mathcal{R}}\} \subseteq [\![1_{\mathcal{S}}]\!].$$

Indeed, Definition 10 simply expresses the bimonoidal functoriality of $[\![-]\!]$, where $\mathcal{S}$ and $\mathcal{P}(\mathcal{R})$ are both considered as bimonoidal categories[6].

▶ **Theorem 11.** *Any interpretation $[\![-]\!]$ of an ordered semi-ring $\mathcal{S}$ into a multiplicity semi-ring $\mathcal{R}$ induces a stratification of the linear category $\boldsymbol{Rel}^{\mathcal{R}}$, defined by:*

$$A^I := \left\{u \in !_{\mathcal{R}} A \mid \sum_{x \in A} u(x) \in [\![I]\!]\right\}, \qquad f^{I \geq J} := \{(u, v) \in !_{\mathcal{R}} f \mid u \in A^I, v \in B^J\},$$

$$\varrho_{I,A} := \{(u, u) \mid u \in A^I\}.$$

In particular, $[\![-]\!]$ extends to a sound interpretation of $\mathrm{B}_{\mathcal{S}}\mathrm{LL}$ *into $\boldsymbol{Rel}^{\mathcal{R}}$.*

Proof. Notice that ϱ is an epi (in fact it is a «surjective» relation) and is natural. Moreover, the morphisms $\mathtt{d}'$, $\mathtt{p}'$, etc... of Definition 6 are obtained by restraining the corresponding $\boldsymbol{Rel}$ morphisms to the domain/codomain, e.g.:

$$\mathtt{c}'_{A,I,J} := \mathtt{c}_A \cap (A^{I+J} \times (A^I \otimes B^J)).$$

One should also prove that these transformations enjoy the diagrams of Figure 3. For example, we should prove that: $\mathtt{c}_A ; (\varrho_{I,A} \otimes \varrho_{J,A}) = \varrho_{I+J,A} ; \mathtt{c}'_{I,J,A}$ (Diagram $\varrho\mathtt{c}$ of Figure 3). Indeed,

$$\mathtt{c}_A ; (\varrho_{I,A} \otimes \varrho_{J,A}) = \{(u + v, (u, v)) \mid \sum_x u(x) \in [\![I]\!], \sum_x v(x) \in [\![J]\!]\}$$
$$\varrho_{I+J,A} ; \mathtt{c}'_{I,J,A} = \{(u + v, (u, v)) \mid \sum_x (u(x) + v(x)) \in [\![I + J]\!], \sum_x u(x) \in [\![I]\!], \sum_x v(x) \in [\![J]\!]\}$$

The two sets are the same because the conditions on u and v imply that on $u + v$, since $[\![I]\!] \oplus [\![J]\!] \subseteq [\![I + J]\!]$. ◀

[6] Actually, $\mathcal{P}(\mathcal{R})$ is a bit less than bimonoidal because just left-distributive.

4.3 Examples

Let us apply Theorem 11 to the ordered semi-rings discussed in Section 2.1.

There is only one possible interpretation of the trivial semi-ring into the multiplicity semi-ring $\mathbb{N}$, associating the unique element $*$ with the whole set $\mathbb{N}$. In fact, Definition 10 requires that $[\![*]\!]$ contains $0, 1$ and that it is closed under addition. This interpretation gives the usual multi-set based model of linear logic. By enlarging the multiplicity semi-ring, for example considering $\overline{\mathbb{N}}$, one can set $[\![*]\!] = \overline{\mathbb{N}}$ and getting the model of linear logic giving rise to the non-sensible models of the untyped λ-calculus studied in [3].

The interpretation of a Boolean-based ordered semi-ring into $\mathbb{N}$ depends on the order between tt and ff. In the case $\mathit{ff} \leq \mathit{tt}$, we can set either $[\![\mathit{tt}]\!] = \mathbb{N}$ and $[\![\mathit{ff}]\!] = \{0\}$, or $[\![\mathit{tt}]\!] = \mathbb{N} = [\![\mathit{ff}]\!]$.[7] The latter collapses the two modalities to the usual multiset comonad, while the former interprets the formula A^{ff} by the singleton of the empty multiset, representing the type of unused resources. In the case ff and tt are incomparable in $\mathcal{S}$, then we can set $[\![\mathit{tt}]\!] = \mathbb{N} - \{0\}$ and $[\![\mathit{ff}]\!] = \{0\}$, strictly distinguishing between used resources of type A^{tt} and unused resources of type A^{ff}.

In the case the syntactic semi-ring $\mathcal{S}$ is already a multiplicity semi-ring (like $\mathbb{N}$, $\mathbb{N}[X_i]_{i\in\mathbb{N}}$ and $\mathbb{N}_f\langle \mathrm{Aff}_1^c \rangle$), then we have a natural interpretation of $\mathcal{S}$ into itself, associating a scalar with the downward closure of its singleton. In fact, we have:

▶ **Proposition 4.** *For any ordered multiplicity semi-ring $(\mathcal{R}, \leq_{\mathcal{R}})$, the following is an interpretation of $\mathcal{R}$ into $\mathcal{R}$:*

$$[\![I]\!] = \{J \mid J \leq_{\mathcal{R}} I\}. \tag{1}$$

Proof. The only condition of Definition 10 which is not so immediate to check is the one dealing with $\odot$. Any element of $[\![I]\!] \odot [\![J]\!]$ is of the form $\sum_i I_i \cdot J_i$ such that $\sum_i J_i \leq_{\mathcal{R}} J$ and for all i, $I_i \leq_{\mathcal{R}} I$. Thus we have:

$$\sum_i I_i \cdot J_i \quad \leq_{\mathcal{R}} \quad \sum_i I \cdot J_i \quad = \quad I \cdot (\sum_i J_i) \quad \leq_{\mathcal{R}} \quad I \cdot J$$

so that $\sum_i I_i \cdot J_i \in [\![I \cdot J]\!]$. ◀

For example, if $\mathcal{R}$ is $\mathbb{N}$, the interpretation of A^n induced by Equation (1) is the set of the multisets with cardinally at most n, if we consider the standard order, or with cardinally exactly n, if we consider the discrete order. In the case of the polynomial semi-ring $(\mathbb{N}[X_i]_{i\in\mathbb{N}}, \times, 1, +, 0)$, Equation (1) associates with $A^{p(x)}$ the set of functions mapping an element $a \in A$ to a polynomial $q(x)$ bounded by $p(x)$ (according to the notion of boundedness described by the order considered).

The interpretation given by Equation (1) faithfully mirrors in the semantics the behavior of the typing system and hence it is uninteresting, at least in our setting. More relevant models can be obtained by shrinking the semantic semi-ring, via the following proposition.

▶ **Proposition 5.** *Given an ordered semi-ring $\mathcal{S}$, an interpretation $[\![-]\!]_{\mathcal{R}}$ of $\mathcal{S}$ into a multiplicity semi-ring $\mathcal{R}$ and a multiplicity sub-semi-ring $\mathcal{R}'$ of $\mathcal{R}$, we have that the map $[\![-]\!]_{\mathcal{R}'} : I \mapsto ([\![I]\!] \cap \mathcal{R}')$ defines an interpretation of $\mathcal{S}$ into $\mathcal{R}'$ whenever it respects the order, i.e.:*

$$I \leq_{\mathcal{S}} J \text{ iff } [\![I]\!]_{\mathcal{R}'} \subseteq [\![J]\!]_{\mathcal{R}'}.$$

[7] There are other uninteresting possibilities.

For example, $\bar{\mathbb{N}}$ can be interpreted into itself by Equation 1, or, by using Proposition 5, into its sub-semi-ring $\mathbb{N}$ by setting $[\![\omega]\!] = \mathbb{N} = [\![\omega]\!]_{\bar{\mathbb{N}}} \cap \mathbb{N}$. This latter interpretation has the virtue of expressing both finite and infinite scalars with sets of finite natural numbers.

If $\mathcal{S}$ is not a multiplicity semi-ring, one can interpret it into the "free" multiplicity semi-ring $\mathbb{N}_f\langle\mathcal{S}.\rangle$ induced by the multiplicative monoid $\mathcal{S}.$ of $\mathcal{S}$ (recall Proposition 3):

▶ **Proposition 6.** *For any ordered semi-ring $(\mathcal{S}, \leq_{\mathcal{S}})$, the following is an interpretation of $\mathcal{S}$ into $\mathbb{N}_f\langle\mathcal{S}.\rangle$ (square brackets $[\ldots]$ below denote standard multisets):*

$$[\![I]\!] = \left\{ [J_1, ..., J_n] \mid \sum_{i \leq n} J_i \leq_{\mathcal{S}} I \right\}. \tag{2}$$

Proof. As in the previous proposition, one has to check (among other equations) that $[\![I]\!] \oplus [\![J]\!] \subseteq [\![I + J]\!]$

$$\begin{aligned} [\![I]\!] \oplus [\![J]\!] &= \{[I_1, ..., I_n, J_1, ..., J_m] \mid \sum_{i \leq n} I_i \leq_S I,\ \sum_{i \leq m} J_i \leq_S J\} \\ &\subseteq \{[J_1, ..., J_n] \mid \sum_{i \leq n} J_i \leq_S I + J\} \\ &= [\![I + J]\!] \end{aligned}$$

◀

This construction makes a sharp difference between the way syntax and semantics express sums between parameters. For example, if you take $\mathcal{S}$ to be $\mathbb{R}^+$ endowed with the usual order, then the interpretation of a type A^r, for $r \in \mathbb{R}^+$, induced by Equation (2) can be seen as the set of finite multisets $[(a_1, r_1), \ldots, (a_n, r_n)]$ of elements in $A \times \mathbb{R}^+$ such that $r_1 + \cdots + r_n \leq r$. The contraction between two types A^r and $A^{r'}$ gives at the level of the syntax the type $A^{r+r'}$ where the two parameters r, r' are totally merged into $r + r'$. While at the level of the semantics we have the set of the disjoint unions of a multiset in A^r and one in $A^{r'}$, so that the real-values r_1, …, r_n are kept distinct.

Such an interpretation of $\mathbb{R}^+$ into $\mathbb{N}_f\langle\mathbb{R}^+.\rangle$ however is not completely satisfactory because it does not express a clear notion of probability. One can get something better by applying Proposition 5. Consider the semi-ring $\mathbb{N}_f\langle[0,1].\rangle$ made by the elements of $\mathbb{N}_f\langle\mathbb{R}^+.\rangle$ that can be seen as multisets of probabilities. Proposition 3 shows that $\mathbb{N}_f\langle[0,1].\rangle$ is a multiplicity semi-ring, and one can easily check that the interpretation $\mathbb{R}^+ \mapsto \mathbb{N}_f\langle\mathbb{R}^+.\rangle$ is still injective when restricted to $\mathbb{N}_f\langle[0,1].\rangle$. So Proposition 5 states that $\boldsymbol{Rel}^{\mathbb{N}_f\langle[0,1].\rangle}$ is a model of $\mathrm{B}_{\mathbb{R}^+}\mathrm{LL}$ by the interpretation:

$$[\![r]\!] := \left\{ [p_1, \ldots, p_n] \mid n \geq 0, p_i \in [0,1], \sum_{i=1}^{n} p_i \leq r \right\}.$$

This interpretation is not only refining the previous one but perfectly fit the intuitive semantics. Indeed, a multiset $[r_1, \ldots, r_n]$ represents n independent calls to a resource, each call answered with a probability $r_i \in [0,1]$. In particular, the expected value of the number of accessible resources is $r_1 + \cdots + r_n \leq r$.

5 Beyond *Rel*

We have seen that the relational category provides a large panel of different semantics, but all of them are definitely degenerated because the ambient category is compact closed. Actually, our tools apply to various other categories, even not compact closed. Just to have a taste of

this generality we discuss here the case of coherence spaces. A more general account will be developed by the first author in his forthcoming Ph.D. thesis.

A coherence space $\mathcal{A}$ is a pair of a set $|\mathcal{A}|$, called *web*, and a reflexive and symmetric relation $\frown_{\mathcal{A}}$, called *coherence*. A coherence space can be seen as a symmetric graph over its web, and in fact we denote by $\boldsymbol{Cl}(\mathcal{A})$ the set of cliques of $\mathcal{A}$, that is $\boldsymbol{Cl}(\mathcal{A}) = \{u \subseteq |\mathcal{A}| \mid \forall a, a' \in u, a \frown_{\mathcal{A}} a'\}$. Given two coherence spaces $\mathcal{A}$, $\mathcal{B}$, the hom-set $\boldsymbol{Coh}(\mathcal{A}, \mathcal{B})$ is the set of relations $r \subseteq |\mathcal{A}| \times |\mathcal{B}|$ such that: for every $(a, b), (a', b') \in r$, if $a \frown_{\mathcal{A}} a'$ then $b \frown_{\mathcal{B}} b'$ and, if moreover $a \neq a'$ then also $b \neq b'$.

Coherence spaces have been introduced by Girard as the first model of linear logic [7]. We omit to give here a detailed description of it, referring to [7] for the details. There are two main linear categories based on coherence spaces, differing on the exponential comonad, one (denoted $\boldsymbol{Coh}_s$) is based on the finite set functor and the other one (denoted by $\boldsymbol{Coh}_m$) on the finite multi-set functor. The action of the two comonads on a coherence space $\mathcal{A}$ is defined as follows ($\mathcal{P}_f$, M_f refer to the set of, respectively, finite sets and multisets):

$$|!_s\mathcal{A}| := \{u \in \mathcal{P}_f(|\mathcal{A}|) \mid u \in \boldsymbol{Cl}(\mathcal{A})\}, \qquad u \frown_{!_s\mathcal{A}} u' := u \cup u' \in \boldsymbol{Cl}(\mathcal{A}),$$
$$|!_m\mathcal{A}| := \{u \in \mathrm{M}_f(|\mathcal{A}|) \mid \mathrm{supp}(u) \in \boldsymbol{Cl}(\mathcal{A})\}, \quad u \frown_{!_m\mathcal{A}} u' := \mathrm{supp}(u) \cup \mathrm{supp}(u') \in \boldsymbol{Cl}(\mathcal{A}).$$

In Section 4.2 we have seen how to define a stratification of $\boldsymbol{Rel}^{\mathcal{R}}$ by interpreting the semi-ring $\mathcal{S}$ into $(\mathcal{P}(\mathcal{R}), \oplus, \odot)$. We chose $(\mathcal{P}(\mathcal{R}), \oplus, \odot)$ because it grows out from an hidden structure of $\boldsymbol{Rel}^{\mathcal{R}}(!_{\mathcal{R}}\mathtt{1}, \mathtt{1})$. In the setting of set-based $\boldsymbol{Coh}_s$, we must consider $\boldsymbol{Coh}_s(!_s\mathtt{1}, \mathtt{1})$ ($\mathtt{1}$ is the one-element coherence space) which gives a three element semi-ring $\mathbb{B}_\perp = (\{\perp, \mathit{ff}, \mathit{tt}\}, \oplus, \odot)$ with $\mathit{ff}, \mathit{tt}, \oplus, \odot$ representing the usual Boolean operations and $\perp$ being absorbing for $\oplus$ and $\mathit{ff} \odot \perp = \mathit{ff}$ (zero case of left-distribution) but $\perp \odot \mathit{ff} = \perp \odot \mathit{tt} = \mathit{tt} \odot \perp = \perp \odot \perp = \perp$. The order is flat: $\perp$ is the bottom element and ff, tt are incomparable. The only interpretation $[\![-]\!] : \mathbb{B}_\perp \mapsto \mathbb{B}_\perp$ respecting the conditions of Definition 10 is the identity function, so that one can recover (by Theorem 11) the stratification defined by:

$$|\mathcal{A}^\perp| := \emptyset, \qquad |\mathcal{A}^{\mathit{ff}}| := \{\emptyset\}, \qquad |\mathcal{A}^{\mathit{tt}}| := \boldsymbol{Cl}(\mathcal{A}) - \{\emptyset\}, \qquad u \frown_{\mathcal{A}^{\mathit{tt}}} u' := u \cup u' \in \boldsymbol{Cl}(\mathcal{A}),$$
$$\varrho_{n,\mathcal{A}} := \{(u, u) \mid u \in |\mathcal{A}^n|\}, \qquad \text{for } n = \perp, 0, 1.$$

In fact, one can easy check that $(\_)^-$ is a bifunctor $\mathbb{B}_\perp \times \boldsymbol{Coh}_s \to \boldsymbol{Coh}_s$, with a natural transformation ϱ satisfying the diagrams of Figure 3.

If you consider the multi-set based linear category $\boldsymbol{Coh}_m$, we have that $\boldsymbol{Coh}_m(!_m\mathtt{1}, \mathtt{1})$ yields $\mathbb{N}_\perp = (\{\perp\} \cup \mathbb{N}, \oplus, \odot)$ with $\oplus, \odot$ the usual sum and product over the natural numbers extended to $\perp$ as in $\mathbb{B}_\perp$. Also in this case the order is flat: $\perp$ is the bottom element and any two non-equal natural numbers are incomparable. If we can consider the identity function as an interpretation $[\![-]\!] : \mathbb{N}_\perp \mapsto \mathbb{N}_\perp$ we get the stratification defined by (where, for $u \in |!_m\mathcal{A}|$, $\#u$ refers to its cardinality as a multiset: $\#u = \sum_{a \in |\mathcal{A}|} u(a)$):

$$|\mathcal{A}^\perp| := \emptyset, \qquad |\mathcal{A}^n| := \{u \in |!_m\mathcal{A}| \mid \#u = n\}, \qquad u \frown_{\mathcal{A}^n} u' := u \frown_{!_m\mathcal{A}} u',$$
$$\varrho_{s,\mathcal{A}} := \{(u, u) \mid u \in |\mathcal{A}^s|\}, \qquad \text{for } s \in \{\perp\} \cup \mathbb{N}.$$

Also in this case, one can easily check that the above is a stratification of $\boldsymbol{Coh}_m$ according to Definition 6.

With a bit of imagination, one can define many other models of $\mathrm{B}_S\mathrm{LL}$ in $\boldsymbol{Coh}_s$ and $\boldsymbol{Coh}_m$ as well as in other linear categories that are not compact closed (e.g. finiteness spaces). Let us stress that the exponential modalities of $\boldsymbol{Rel}$, $\boldsymbol{Coh}_s$, $\boldsymbol{Coh}_m$ and that of finiteness spaces, for example, are not trivial instances of a simple common construction (see [11] for an interesting discussion on this matter). These examples then show the relevance of the notion of stratification in a rather wide class of ambient categories.

6 Conclusion

Full Linear Logic. $\mathrm{B}_{\mathcal{S}}\mathrm{LL}$ is a refinement of the multiplicative exponential fragment of intuitionistic linear logic. One can wonder whether this approach can be extended to full linear logic, with additive connectives and involutive negation.

Additive connectives can be introduced without any difficulty, but the involutive negation, and especially the introduction of the why-not modality ? dual of the of-course !, is more delicate. Namely, one should grasp the computational meaning of the action of the $\mathcal{S}$ parameters over the why-not modality.

Toward true dependent types. The major weakness of $\mathrm{B}_{\mathcal{S}}\mathrm{LL}$ is the lack of dependent types, in particular $\mathrm{B}_{\mathcal{S}}\mathrm{LL}$ is not an extension of bounded linear logic. The interest in this latter has been recently renewed by a series of work, like Dal Lago and Gaboardi's $\mathrm{D}_{\ell}\mathrm{PCF}$ [4] or Gaboardi et al.'s DFuzz [5]. These systems use parameters depending on variables which can be bounded and instantiated in the type derivation. This allows, for example, to distinguish between the resource usages of two branches of a conditional, or, combined with a fix-point combinator, to define a parameter depending on the number of loops performed during the evaluation of an iteration.

It is not clear to us whether and how our semantics extends to such a framework. The notion of dependence is delicate to catch semantically, namely it amounts to making the operator $\mathcal{O}$ parametrised by some context. We are, in fact, currently investigating in a different approach (that still use the same intuitions).

In this approach we do not start with a categorical model of linear logic, but with a richer 2-categorical model of linear logic (for example ***Rel*** endowed with inclusions as 2-functors). Rather than having to find manually a stratification, we can directly identify a structure of $\mathrm{B}_{\mathcal{S}}\mathrm{LL}$ in the lax slice category $\mathcal{C}_{/\mathbb{1}}$ (with morphisms of $\mathcal{C}$ targeting $\mathbb{1}$ as objects). There, the syntactic semiring $\mathcal{S}$ is modeled by a sub category of $\mathcal{C}[!\mathbb{1}, \mathbb{1}]$ (with arrows representing the order relation). This extends naturally to dependency by considering every lax slice category $\mathcal{C}_{/A}$ together, for duplicable objects $A \in \mathcal{C}$ representing the values we are dependent on.

Beyond *Rel*. This paper is focused on the relational category ***Rel*** and on the notion of stratification (Definition 6). This was actually the original goal of our investigation: constructing relational models of $\mathrm{B}_{\mathcal{S}}\mathrm{LL}$. Indeed, it is clear that a more general principle comes out from our results, relating the stratification to a semi-ring structure hidden behind the hom-sets $\mathcal{C}(!\mathbb{1}, \mathbb{1})$.

We have briefly discussed such a generality in Section 5, giving examples of stratifications of linear categories based on coherence spaces. The present setting however cannot explain how one can recover the target semi-ring of an interpretation out of $\mathcal{C}(!\mathbb{1}, \mathbb{1})$, for any (or a large class of) linear category $\mathcal{C}$. To do that one should work in the framework of the 2-categorical models of linear logic, as mentioned in the previous paragraph, and this will be done in the future.

References

1 Nick Benton, Gavin Bierman, Valeria de Paiva, and Martin Hyland. Linear lambda-calculus and categorical models revisited. In E. Börger, G. Jäger, H. Kleine Büning, S. Martini, and M. Richter, editors, *Proceedings of the Sixth Workshop on Computer Science Logic*, pages 61–84. Springer Verlag, 1993.

2 Aloïs Brunel, Marco Gaboardi, Damiano Mazza, and Steve Zdancewic. A core quantitative coeffect calculus. In *Proceedings of ESOP 2014*, volume 8410 of *LNCS*, pages 351–370. Springer, 2014.

3 Alberto Carraro, Thomas Ehrhard, and Antonino Salibra. Exponentials with infinite multiplicities. In A. Dawar and H. Veith, editors, *Proceedings of CSL 2010*, volume 6247 of *LNCS*, pages 170–184, 2010.

4 Ugo Dal Lago and Marco Gaboardi. Linear dependent types and relative completeness. In *Proceedings of LICS 2011*, pages 133–142. IEEE, 2011.

5 Marco Gaboardi, Andreas Haeberlen, Justin Hsu, Arjun Narayan, and Benjamin C. Pierce. Linear dependent types for differential privacy. In R. Giacobazzi and R. Cousot, editors, *Proceedings of POPL 2013*, pages 357–370. ACM, 2013.

6 Dan R. Ghica and Alex I. Smith. Bounded linear types in a resource semiring. In Z. Shao, editor, *Proceedings of ESOP 2014*, volume 8410 of *LNCS*, pages 331–350. Springer, 2014.

7 Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

8 Jean-Yves Girard, Andre Scedrov, and Philip J. Scott. Bounded linear logic: A modular approach to polynomial-time computability. *Theor. Comput. Sci.*, 97(1):1–66, April 1992.

9 Charles Grellois and Paul-André Melliès. An infinitary model of linear logic. In *Proceedings of FoSSaCS 2015*, volume 9034 of *LNCS*, pages 41–55. Springer, 2015.

10 Paul-Andre Mellies. The parametric continuation monad. *Mathematical Structures in Computer Science*, Festschrift in honor of Corrado Böhm for his 90th birthday, 2013.

11 Paul-André Melliès, Nicolas Tabareau, and Christine Tasson. An explicit formula for the free exponential modality of linear logic. In S. Albers, A. Marchetti-Spaccamela, Y. Matias, S. E. Nikoletseas, and W. Thomas, editors, *Proceedings of ICALP 2009*, pages 247–260, 2009.

12 Tomas Petricek, Dominic Orchard, and Alan Mycroft. Coeffects: unified static analysis of context-dependence. In *Proceedings of International Conference on Automata, Languages, and Programming – Volume Part II*, ICALP, 2013.

13 Tomas Petricek, Dominic Orchard, and Alan Mycroft. Coeffects: A calculus of context-dependent computation. In *Proceedings of International Conference on Functional Programming*, ICFP, 2014.

On Classical PCF, Linear Logic and the MIX Rule*

Shahin Amini[1] and Thomas Ehrhard[2]

1 **PPS, UMR 7126, Université Paris Diderot, Sorbonne Paris Cité**
F-75205 Paris, France
shahin.a.amini@gmail.com

2 **CNRS, PPS, UMR 7126, Université Paris Diderot, Sorbonne Paris Cité**
F-75205 Paris, France
thomas.ehrhard@pps.univ-paris-diderot.fr

Abstract

We study a classical version of PCF from a semantic point of view. We define a general notion of model based on categorical models of Linear Logic, in the spirit of earlier work by Girard, Regnier and Laurent. We give a concrete example based on the relational model of Linear Logic, that we present as a non-idempotent intersection type system, and we prove an Adequacy Theorem using ideas introduced by Krivine. Following Danos and Krivine, we also consider an extension of this language with a MIX construction introducing a form of must non-determinism; in this language, a program of type integer can have more than one value (or no value at all, raising an error). We propose a refinement of the relational model of classical PCF in which programs of type integer are single valued; this model rejects the MIX syntactical constructs (and the MIX rule of Linear Logic).

1998 ACM Subject Classification F.3.2 Semantics of Programming Languages, F.3.3 Studies of Program Constructs, F.4.1 Mathematical Logic

Keywords and phrases lambda-calculus, linear logic, classical logic, denotational semantics

Digital Object Identifier 10.4230/LIPIcs.CSL.2015.582

1 Introduction

Since the fundamental discovery by Timothy Griffin [7] that the call/cc primitive of the programming language *Scheme* can be typed by a non intuitionistic classical tautology (the Law of Peirce), much work has been dedicated to the study of the computational content of classical proofs. A particular attention has been devoted to the denotational semantics of classically extended lambda-calculi. Among the notions of categorical model defined for this purpose, we can isolate two main concepts.

- Models extending the standard categorical setting of CCC's for interpreting usual λ-calculi through a CPS translation. In these models of classical λ-calculi, terms are interpreted in a CCC of *negated objects*, that is objects of shape Σ^P where P is an object of a cartesian and cocartesian category $\mathcal{P}$ where Σ is a distinguished baseable object, see [15] and its generalization [14] where the category of negated objects is axiomatized. In this latter paper, it is also shown that Parigot's $\lambda\mu$-calculus [13] is the internal language of such categories and completeness of this notion of model for $\lambda\mu$ theories is proven in [9], enforcing further its universality.

* This work was partially supported by the French-Chinese ANR-NSFC project Locali.

- Models based on Linear Logic (LL) and polarities. The basic idea of such models is to divide objects (formulas) in two categories, exchanged by linear negation: negative objects and positive ones: this is the basic idea of Girard's LC logical system. The main feature of these polarized objects is that each one carries its own structural morphisms (weakening and contraction). In [5], positive objects are correlation spaces (commutative $\otimes$-comonoids), and it is crucially used that all such comonoids are coalgebras for the $!\_$ functor, because, in the considered coherence space model, this exponential is the free commutative $\otimes$-comonoid functor. An obvious generalization, implicitly considered in [11], is to interpret directly positive formulas as !-coalgebras instead of $\otimes$-comonoids without making further assumptions on the ! comonad.

As recalled in Section 3.2, all models of the second class can be seen as models of the first one, but it is not clear that such a presentation is always particularly enlightening. We rather believe that, depending on the considered concrete model, one presentation might be more convenient than the other; for example, the classical PCF game model of [8] and the polarized HO game model of [10] are suitably described using the first notion. In the present paper, we focus on models for which the second presentation is more convenient.

To define the general interpretation of classical PCF, we assume therefore to be given a categorical model of classical LL $\mathcal{L}$ with a few additional features: an object N for natural numbers which is the coproduct of ω copies of 1 (the tensor unit) in $\mathcal{L}$ as well as a fix-point operator at each object, in the Kleisli category of !, which is a CCC.

Our classical version of PCF is based on the $\overline{\lambda}\mu$-calculus described in [1] which features three kinds of expressions: terms, stacks (or continuations) and commands which are pairings of terms and stacks. The operational semantics is given as a rewriting system on commands (it can easily be extended to terms and stacks but we do not do it for lack of space).

Stacks can be duplicated or erased during computations, hence types are interpreted as !-coalgebras and stacks as coalgebra morphisms. Notice that the interpretation of types corresponds to the linear negation of the usual PCF interpretation of types: roughly speaking, we interpret $\sigma \Rightarrow \tau$ as $!(P^\perp) \otimes Q$ where P and Q are the interpretations of σ and τ ($P^\perp$ is the linear negation of P). We retrieve the ordinary interpretation simply by taking the linear negation of this *positive translation*[1].

In particular, the positive interpretation of the ground type of natural numbers has to be such a coalgebra. Therefore, the most tempting choice, which would be to take $[\![\iota]\!] = \mathsf{N}^\perp$, is not possible (*Warning:* N is canonically a !-coalgebra, but $[\![\iota]\!] = \mathsf{N}^\perp$ is not!). So we simply set $[\![\iota]\!] = !(\mathsf{N}^\perp)$, the free !-coalgebra generated by $\mathsf{N}^\perp$. We do not know if, depending on the concrete model under consideration, more "economical" choices of !-coalgebras would have been possible; this is certainly an interesting research direction. We describe the corresponding interpretation of expressions and state a general soundness theorem: this interpretation of commands is invariant under reduction (of course this could be extended to terms and stacks in a setting where the reduction would be extended to these expressions).

Then we consider the simplest example of this situation, where we take for $\mathcal{L}$ the category of sets and relations, which is a well known model of LL. We provide a description of the interpretation of expressions in this particular model by means of an intersection typing

[1] Due to the symmetries of a categorical model of LL, this is more an aesthetic choice of design than anything else. We could have preferred a negative interpretation, representing stacks as ?-algebras and using $\parr$ instead of $\otimes$ for interpreting contexts. The two interpretations would have been the same, up to linear transposition. The positive interpretation is in some sense closer to usual λ-calculus intuitions because, when interpreting expressions, the context remains on the argument side of morphisms.

system, in the spirit of [3, 16]. We then prove an adequacy theorem: if a command has a non-empty interpretation (that is, if it is typable in this intersection typing system) then its reduction terminates. The proof is based on a standard reducibility method (see [2] for instance). This model accommodates a natural extension of classical PCF by a parallel composition of commands corresponding to the MIX rule of LL as in [2].

In classical PCF with MIX, a normalizing command without free variables but with free names of ground type ι can yield an arbitrary amount of unrelated natural numbers on each of its free names (outputs). Without MIX syntactical constructs such a command will produce exactly one natural number on exactly one of its outputs. This can be checked syntactically, but we also build a simple refinement of the relational model of LL which does not accommodate the MIX rule and gives a direct semantic account of this uniqueness of values for classical PCF without MIX. This means that this crucial property will remain true in any extension of classical PCF that can be interpreted in this model.

2 Classical PCF

Types are given by the following BNF syntax: $\sigma := \iota \mid \sigma \Rightarrow \sigma \mid \sigma \times \sigma$.

The expressions of our language are those of the $\overline{\lambda}\mu$-calculus [1], extended with fix-points and primitives for dealing with integers. Let $x, y \ldots$ be variables and $\alpha, \beta \ldots$ be names. Terms t, commands c and stacks π are defined as follows (with $n \in \mathbb{N}$):

$$t := x \mid \underline{n} \mid \lambda x^\sigma\, t \mid \langle t, t\rangle \mid \mu\alpha^\sigma\, c \mid \mathsf{fix}\, x^\sigma\, t \qquad c := t * \pi$$
$$\pi := \alpha \mid \mathsf{arg}(t)\cdot\pi \mid \mathsf{pr}_1\cdot\pi \mid \mathsf{pr}_2\cdot\pi \mid \mathsf{succ}\cdot\pi \mid \mathsf{pred}\cdot\pi \mid \mathsf{if}(t,t)\cdot\pi$$

We give now the typing rules, which correspond to a sequent calculus. Γ's are typing variable contexts and Δ's are typing name contexts. We give rules for term typing judgments $\Gamma \vdash t : \sigma \mid \Delta$, stack typing judgments $\Gamma \mid \pi : \sigma \vdash \Delta$ and command typing judgments $\Gamma \vdash c \mid \Delta$.

$$\frac{}{\Gamma, x:\sigma \vdash x : \sigma \mid \Delta} \qquad \frac{}{\Gamma \mid \alpha : \sigma \vdash \alpha : \sigma, \Delta} \qquad \frac{\Gamma \vdash t : \sigma \mid \Delta \qquad \Gamma \mid \pi : \sigma \vdash \Delta}{\Gamma \vdash t * \pi \mid \Delta}$$

$$\frac{\Gamma \vdash s : \sigma \mid \Delta \qquad \Gamma \vdash t : \tau \mid \Delta}{\Gamma \vdash \langle s, t\rangle : \sigma \times \tau \mid \Delta} \qquad \frac{\Gamma \mid \pi : \sigma \vdash \Delta}{\Gamma \mid \mathsf{pr}_1\cdot\pi : \sigma\times\tau \vdash \Delta} \qquad \frac{\Gamma \mid \pi : \tau \vdash \Delta}{\Gamma \mid \mathsf{pr}_2\cdot\pi : \sigma\times\tau \vdash \Delta}$$

$$\frac{\Gamma, x:\sigma \vdash t : \tau \mid \Delta}{\Gamma \vdash \lambda x^\sigma\, t : \sigma \Rightarrow \tau \mid \Delta} \qquad \frac{\Gamma \vdash t : \sigma \mid \Delta \qquad \Gamma \mid \pi : \tau \vdash \Delta}{\Gamma \mid \mathsf{arg}(t)\cdot\pi : \sigma \Rightarrow \tau \vdash \Delta} \qquad \frac{\Gamma \vdash c \mid \alpha : \sigma, \Delta}{\Gamma \vdash \mu\alpha^\sigma\, c : \sigma \mid \Delta}$$

$$\frac{}{\Gamma \vdash \underline{n} : \iota \mid \Delta} \qquad \frac{\Gamma \mid \pi : \iota \vdash \Delta}{\Gamma \mid \mathsf{succ}\cdot\pi : \iota \vdash \Delta} \qquad \frac{\Gamma \mid \pi : \iota \vdash \Delta}{\Gamma \mid \mathsf{pred}\cdot\pi : \iota \vdash \Delta}$$

$$\frac{\Gamma \vdash t_1 : \sigma \mid \Delta \qquad \Gamma \vdash t_2 : \sigma \mid \Delta \qquad \Gamma \mid \pi : \sigma \vdash \Delta}{\Gamma \mid \mathsf{if}(t_1,t_2)\cdot\pi : \iota \vdash \Delta} \qquad \frac{\Gamma, x:\sigma \vdash t : \sigma \mid \Delta}{\Gamma \vdash \mathsf{fix}\, x^\sigma\, t : \sigma \mid \Delta}$$

We define a deterministic reduction relation $\to$ on processes.

$$(\lambda x^\sigma\, s) * \mathsf{arg}(t)\cdot\pi \to s\,[t/x] * \pi \qquad \langle s, t\rangle * \mathsf{pr}_1\cdot\pi \to s * \pi \qquad \langle s, t\rangle * \mathsf{pr}_2\cdot\pi \to t * \pi$$
$$(\mu\alpha^\sigma\, c) * \pi \to c\,[\pi/\alpha] \qquad (\mathsf{fix}\, x^\sigma\, t) * \pi \to t\,[\mathsf{fix}\, x^\sigma\, t/x] * \pi \qquad \underline{n} * \mathsf{succ}\cdot\pi \to \underline{n+1} * \pi$$
$$\underline{0} * \mathsf{pred}\cdot\pi \to \underline{0} * \pi \qquad \underline{n+1} * \mathsf{pred}\cdot\pi \to \underline{n} * \pi$$
$$\underline{0} * \mathsf{if}(t_1,t_2)\cdot\pi \to t_1 * \pi \qquad \underline{n+1} * \mathsf{if}(t_1,t_2)\cdot\pi \to t_2 * \pi$$

▶ **Proposition 1** (Subject Reduction). *Assume that* $\Gamma \vdash c \mid \Delta$ *and* $c \to c'$. *Then* $\Gamma \vdash c' \mid \Delta$.

The proof is a straightforward case analysis involving a Substitution Lemma.

A typical example of classical PCF program is the call/cc operator $t = \lambda f^{(\iota\Rightarrow\sigma)\Rightarrow\iota}\,\mu\alpha^{\iota}\,(f * \mathsf{arg}(\lambda x^{\iota}\,\mu\beta^{\sigma}\,(x * \alpha)) \cdot \alpha)$ which satisfies $\vdash t : ((\iota \Rightarrow \sigma) \Rightarrow \iota) \Rightarrow \iota \mid$ (its type is an instance of the well known Peirce classical tautology). When fed with an argument s such that $\vdash s : (\iota \Rightarrow \sigma) \Rightarrow \iota \mid$ (a functional), t tests whether this functional returns directly a value (and then t returns that value) or uses its argument (a function) by providing it with a natural number $\underline{n}$, and in that case t returns $\underline{n}$. This choice between two options is implemented by a contraction (the two occurrences of α).

The MIX extension. We consider also an extension of this classical version of PCF where we add two new constructs: a command err and, given commands c and d, a command $c\|d$. A similar extension of an untyped classical calculus has already been considered in [2]. It is more naturally introduced at the level of commands in the present $\overline{\lambda}\mu$ setting. These constructions obey the following typing rules

$$\frac{}{\Gamma \vdash \mathsf{err} \mid \Delta} \qquad \frac{\Gamma \vdash c \mid \Delta \qquad \Gamma \vdash d \mid \Delta}{\Gamma \vdash (c\|d) \mid \Delta}$$

We then extend the operational semantics of the calculus by adding the following reduction rules for commands.

$$\frac{}{\mathsf{err}\|c \to c} \qquad \frac{}{c\|\mathsf{err} \to c} \qquad \frac{c \to c'}{c\|d \to c'\|d} \qquad \frac{d \to d'}{c\|d \to c\|d'}$$

The resulting calculus on commands (with the other reduction rules given in Section 2) clearly satisfies the diamond property, the strongest form of confluence. These constructions can be extended as term constructions, available at all types. Simply set $\mathsf{err}^{\sigma} = \mu\alpha^{\sigma}\,\mathsf{err}$ and $(s\|t) = \mu\alpha^{\sigma}\,(s * \alpha\|t * \alpha)$. The term $s\|t$ is as a parallel composition of s and t enriching the language with a form of *must non-determinism*. It allows *eg.* to write $\underline{3}\|\underline{7}$, a closed term of type ι, whose value is *at the same time* 3 and 7.

Almost closed commands. We come back to our initial version of classical PCF, without the MIX constructs. A name context $\Delta = (\alpha_1 : \tau_1, \dots, \alpha_k : \tau_k)$ is *ground* if $\tau_j = \iota$ for each j. We say that a command c is *almost closed* if $\vdash c \mid \Delta$ for some ground Δ. An almost closed command is very similar to a closed term of type ι in ordinary PCF. The difference is twofold: first an almost closed command can have more than one output (one for each name in the name context), and second its outputs are named, simply to make them usable.

▶ **Proposition 2.** *Let c be an almost closed and normal command. Then $c = \underline{n} * \alpha$ for some $n \in \mathbb{N}$ and name α.*

The proof is a simple case analysis.

So, consider an almost closed command c such that, say, $\vdash c \mid \alpha_1 : \iota, \dots, \alpha_k : \iota$. Then either the $\to$ reduction of c does not terminate, or it ends with a normal almost closed command, which must be of shape $\underline{n} * \alpha_i$ for uniquely determined $i \in \{1, \dots, k\}$ and $n \in \mathbb{N}$: the reduction of c computes the value $\underline{n}$ and chooses the output on which it is issued.

The notion of almost closed command still makes sense in classical PCF with MIX. The difference is that normal forms are now MIX compositions of elementary command $\underline{n} * \alpha_i$. One can obtain for instance $(\underline{0} * \alpha_1)\|((\underline{3} * \alpha_2)\|(\underline{7} * \alpha_1))$ whose effect is to produce the value $\underline{3}$ on output α_2, values $\underline{0}$ and $\underline{7}$ on output α_1 and nothing on the other outputs.

3 Linear logic based denotational semantics

The kind of denotational models in which we are interested in this paper are those induced by a model of LL, in the spirit of Girard's seminal work [5] further developed *eg.*. in [11]. We first recall the general definition of a model of LL implicit in [4], our main reference here is [12] to which we also refer for the rich bibliography on this general topic. A model of LL consists of:

- A category $\mathcal{L}$.
- A symmetric monoidal closed structure $(\otimes, 1, \lambda, \rho, \alpha, \sigma)$: $\otimes$ is a functor $\mathcal{L}^2 \to \mathcal{L}$, 1 an object of $\mathcal{L}$, $\lambda_X \in \mathcal{L}(1 \otimes X, X)$, $\rho_X \in \mathcal{L}(X \otimes 1, X)$, $\alpha_{X,Y,Z} \in \mathcal{L}((X \otimes Y) \otimes Z, X \otimes (Y \otimes Z))$ and $\sigma_{X,Y} \in \mathcal{L}(X \otimes Y, Y \otimes X)$ are natural isos satisfying coherence diagrams that we do not recall here. We use $X \multimap Y$ for the object of linear morphisms from X to Y, ev for the evaluation morphism which belongs to $\mathcal{L}((X \multimap Y) \otimes X, Y)$ and cur for the map $\mathcal{L}(Z \otimes X, Y) \to \mathcal{L}(Z, X \multimap Y)$.
- An object $\bot$ of $\mathcal{L}$ such that $\eta_X = \mathsf{cur}(\mathsf{ev}\ \sigma_{X \multimap \bot, X}) \in \mathcal{L}(X, (X \multimap \bot) \multimap \bot)$ be an iso for each object X (one says that $\mathcal{L}$ is a $*$-autonomous category); we use $X^\bot$ for $X \multimap \bot$.
- The category $\mathcal{L}$ is assumed to be cartesian. We use $\top$ for the terminal object, & for the cartesian product and pr^i for the projections. It follows by $*$-autonomy that $\mathcal{L}$ has also all finite coproducts.
- We are also given a comonad $!\_ : \mathcal{L} \to \mathcal{L}$ with counit $\mathsf{der}_X \in \mathcal{L}(!X, X)$ (called dereliction) and comultiplication $\mathsf{dig}_X \in \mathcal{L}(!X, !!X)$ (called digging).
- And a strong symmetric monoidal structure for the functor $!\_$, from the symmetric monoidal category $(\mathcal{L}, \&)$ to the symmetric monoidal category $(\mathcal{L}, \otimes)$. This means that we are given an iso $\mathsf{m}^{(0)} \in \mathcal{L}(1, !\top)$ and a natural iso $\mathsf{m}^{(2)}_{X,Y} \in \mathcal{L}(!X \otimes !Y, !(X \,\&\, Y))$ which satisfy a series of commutations that we do not recall here (they are often called *Seely isos*). We also require a coherence condition relating $\mathsf{m}^{(2)}$ and dig.

It follows that we can define a lax symmetric monoidal structure for the functor $!\_$ from the symmetric monoidal category $(\mathcal{L}, \otimes)$ to itself, that is a natural morphism $\mu^{(n)}_{X_1,\dots,X_N} \in \mathcal{L}(!X_1 \otimes \cdots \otimes !X_n, !(X_1 \otimes \cdots \otimes X_n))$ satisfying some coherence conditions.

We use $?\_$ for the "De Morgan dual" of $!\_$: $?X = (!(X^\bot))^\bot$ and similarly for morphisms. It is a monad on $\mathcal{L}$ with unit der'_X and multiplication dig'_X defined straightforwardly, using der_Y and dig_Y.

The Eilenberg-Moore category. It is then standard to define the category $\mathcal{L}^!$ of !-coalgebras. An object of this category is a pair $P = (\underline{P}, \mathsf{h}_P)$ where $\underline{P} \in \mathsf{Obj}(\mathcal{L})$ and $\mathsf{h}_P \in \mathcal{L}(\underline{P}, !\underline{P})$ is such that $\mathsf{der}_{\underline{P}}\ \mathsf{h}_P = \mathsf{Id}$ and $\mathsf{dig}_{\underline{P}}\ \mathsf{h}_P = !\mathsf{h}_P\ \mathsf{h}_P$.

Given two such coalgebras P and Q, an element of $\mathcal{L}^!(P, Q)$ is an $f \in \mathcal{L}(\underline{P}, \underline{Q})$ such that $\mathsf{h}_Q\ f = !f\ \mathsf{h}_P$. Identities and composition are defined in the obvious way. The functor $!\_$ can then be seen as a functor from $\mathcal{L}$ to $\mathcal{L}^!$: this functor maps X to the coalgebra $(!X, \mathsf{dig}_X)$ and a morphism $f \in \mathcal{L}(X, Y)$ to the coalgebra morphism $!f \in \mathcal{L}^!((!X, \mathsf{dig}_X), (!Y, \mathsf{dig}_Y))$. It is right adjoint to the forgetful functor $\mathsf{U} : \mathcal{L}^! \to \mathcal{L}$ which maps a !-coalgebra P to $\underline{P}$ and a morphism f to itself. Given $f \in \mathcal{L}(\underline{P}, X)$ (where X is an object of $\mathcal{L}$ and P an object of $\mathcal{L}^!$), we use $f^!$ for the corresponding element of $\mathcal{L}^!(P, !X)$, called *generalized promotion* of f.

The object 1 of $\mathcal{L}$ induces an object of $\mathcal{L}^!$, still denoted as 1, namely $(1, \mu^{(0)})$.

Given two objects P and Q of $\mathcal{L}^!$, we can define an object $P \otimes Q$ of $\mathcal{L}^!$ setting $\underline{P \otimes Q} = \underline{P} \otimes \underline{Q}$ and $\mathsf{h}_{P \otimes Q} = \mu^{(2)}_{\underline{P},\underline{Q}}\ (\mathsf{h}_P \otimes \mathsf{h}_Q)$.

Any object P of $\mathcal{L}^!$ can be equipped with a canonical structure of commutative comonoid. This means that we can define a morphism $\mathsf{w}_P \in \mathcal{L}^!(P, 1)$ and a morphism $\mathsf{c}_P \in \mathcal{L}^!(P, P \otimes P)$

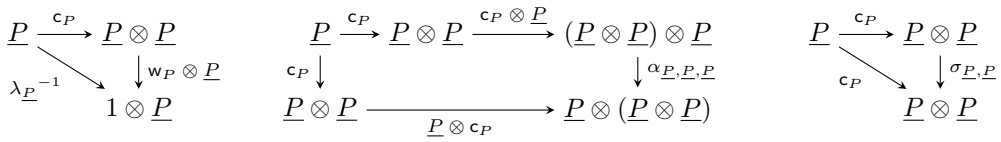


Figure 1 Comonoid properties of a coalgebra.

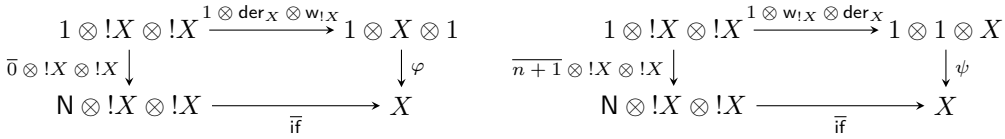


Figure 2 Categorical properties of the conditional.

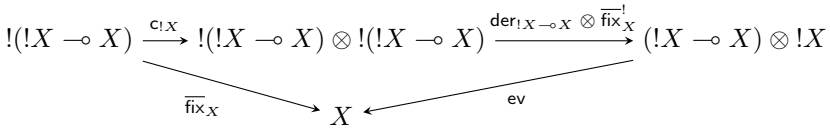
which satisfy the commutations of Figure 1.

One can check a stronger property, namely that 1 is the terminal object of $\mathcal{L}^!$ and that $P \otimes Q$ (equipped with projections defined in the obvious way using w_Q and w_P) is the cartesian product of P and Q in $\mathcal{L}^!$; the proof consists of rather long computations, see [12].

It is also important to notice that, if the family $(P_i)_{i\in I}$ of objects of $\mathcal{L}^!$ is such that the family $(\underline{P_i})_{i\in I}$ admits a coproduct $(\bigoplus_{i\in I}\underline{P_i}, (\mathsf{in}^i)_{i\in I})$ in $\mathcal{L}$, then it admits a coproduct in $\mathcal{L}^!$. This coproduct $P = \bigoplus_{i\in I} P_i$ is defined as $\underline{P} = \bigoplus_{i\in I}\underline{P_i}$, with a structure map h_P defined by the fact that, for each $i \in I$, $\mathsf{h}_P\,\mathsf{in}^i = !\mathsf{in}^i\,\mathsf{h}_{P_i}$.

Object of natural numbers and conditional. We assume also that in $\mathcal{L}$, the family of objects $(X_n)_{n\in\mathbb{N}}$ such that $X_n = 1$ for each n, has a coproduct N. For each $n \in \mathbb{N}$, we use $\overline{n}$ for the nth injection $\overline{n} \in \mathcal{L}(1, \mathsf{N})$. Using the obvious iso between N and $1 \oplus \mathsf{N}$, we define two morphisms $\overline{\mathsf{succ}}, \overline{\mathsf{pred}} \in \mathcal{L}(\mathsf{N}, \mathsf{N})$ such that $\overline{\mathsf{succ}}\,\overline{n} = \overline{n+1}$, $\overline{\mathsf{pred}}\,\overline{0} = \overline{0}$ and $\overline{\mathsf{pred}}\,\overline{n+1} = \overline{n}$. Let X be an object of $\mathcal{L}$. Let $\overline{\mathsf{if}}_0 \in \mathcal{L}(1 \otimes !X \otimes !X, X)$ be defined as the following composition in $\mathcal{L}$: $1 \otimes !X \otimes !X \xrightarrow{1 \otimes \mathsf{der}_X \otimes \mathsf{w}_{!X}} 1 \otimes X \otimes 1 \xrightarrow{\varphi} X$, where φ is the obvious iso. Let $\overline{\mathsf{if}}_+ \in \mathcal{L}(\mathsf{N} \otimes !X \otimes !X, X)$ be defined as the following composition of morphisms in $\mathcal{L}$: $\mathsf{N} \otimes !X \otimes !X \xrightarrow{\mathsf{w}_\mathsf{N} \otimes \mathsf{w}_{!X} \otimes \mathsf{der}_X} 1 \otimes 1 \otimes X \xrightarrow{\psi} X$, where ψ is the obvious iso. Observe that we use the fact that N has a canonical structure of !-coalgebra (as a sum of coalgebras) inducing the weakening morphism w_N. It is the only place where this property is used. Using these two morphisms, the iso between N and $1 \oplus \mathsf{N}$ and the fact that $\otimes$ commutes with sums (because it is a left adjoint), we define a morphism $\overline{\mathsf{if}} \in \mathcal{L}(\mathsf{N} \otimes !X \otimes !X, X)$ such that the two diagrams of Figure 2 commute.

Fix-point operators. For any object X, we assume to be given a morphism $\overline{\mathsf{fix}}_X \in \mathcal{L}(!(!X \multimap X), X)$ such that the following diagram commutes in $\mathcal{L}$:



MIX in Linear Logic The categorical setting introduced so far allows to interpret the MIX-free version of classical PCF. In order to interpret the MIX extension of Section 2, it suffices to assume that $\bot$ is equipped with a structure of commutative $\otimes$-monoid (this is an additional structure of the model). If we think of $\bot$ as an object of scalars, which is a natural

intuition since $\bot$ is the dualizing object, this means that these scalars have a multiplication, a natural intuition again if we have linear algebra in mind. We use $\mathsf{mix}^0 \in \mathcal{L}(1,\bot)$ for the unit of this monoid and $\mathsf{mix}^2 \in \mathcal{L}(\bot\otimes\bot,\bot)$ for its multiplication. When this structure is added, we say that $\mathcal{L}$ is a model of LL with MIX.

3.1 Interpreting classical PCF

The semantics of a type σ is an object $[\![\sigma]\!]$ of $\mathcal{L}^!$. We set $[\![\iota]\!] = !(\mathsf{N}^\bot)$, $[\![\sigma\Rightarrow\tau]\!] = !(\underline{[\![\sigma]\!]}^\bot)\otimes [\![\tau]\!]$ and $[\![\sigma\times\tau]\!] = [\![\sigma]\!]\oplus[\![\tau]\!]$. So $\underline{[\![\iota]\!]}^\bot = ?\mathsf{N}$, which will be the target object for the interpretation of terms of type ι. Let $\Gamma = (x_1:\sigma_1,\dots,x_n:\sigma_n)$ be a variable context and $\Delta = (\alpha_1:\tau_1,\dots,\alpha_k;\tau_k)$ be a name context, then we define two objects of $\mathcal{L}^!$ by $[\![\Gamma]\!] = !(\underline{[\![\sigma_1]\!]}^\bot)\otimes\cdots\otimes !(\underline{[\![\sigma_n]\!]}^\bot)$ and $[\![\Delta]\!] = [\![\tau_1]\!]\otimes\cdots\otimes[\![\tau_k]\!]$. With any term t such that $\Gamma\vdash t:\sigma\mid\Delta$, we associate $[\![t]\!]_{\Gamma,\Delta}\in\mathcal{L}([\![\Gamma]\!]\otimes\underline{[\![\Delta]\!]},\underline{[\![\sigma]\!]}^\bot)$, with any command c such that $\Gamma\vdash c\mid\Delta$ we associate $[\![c]\!]_{\Gamma,\Delta}\in\mathcal{L}([\![\Gamma]\!]\otimes\underline{[\![\Delta]\!]},\bot)$ and with any stack π such that $\Gamma\mid\pi:\sigma\vdash\Delta$ we associate $[\![\pi]\!]_{\Gamma,\Delta}\in\mathcal{L}^!([\![\Gamma]\!]\otimes[\![\Delta]\!],[\![\sigma]\!])$. This latter has to be a coalgebra morphism because stacks must be duplicable and discardable (think of the reduction rule $(\mu\alpha\, c)*\pi\to c\,[\pi/\alpha]$).

We give now the interpretation of expressions, starting with terms. In these definitions, the symbol φ stands for an iso which can be deduced from the context. The interpretation of a variable $[\![x]\!]_{\Gamma,x:\sigma,\Delta}$ is defined as the following composition of morphisms:

$$\underline{[\![\Gamma]\!]}\otimes !(\underline{[\![\sigma]\!]}^\bot)\otimes\underline{[\![\Delta]\!]}\xrightarrow{\mathsf{w}_{[\![\Gamma]\!]}\otimes\mathsf{der}_{\underline{[\![\sigma]\!]}^\bot}\otimes\mathsf{w}_{[\![\Delta]\!]}} 1\otimes\underline{[\![\sigma]\!]}^\bot\otimes 1\xrightarrow{\varphi}\underline{[\![\sigma]\!]}^\bot$$

Let $n\in\mathbb{N}$, remember that $\overline{n}\in\mathcal{L}(1,\mathsf{N})$ so that $?\overline{n}\in\mathcal{L}(?1,?\mathsf{N})$. We define $[\![\underline{n}]\!]_{\Gamma,\Delta}$ as the following composition of morphisms in $\mathcal{L}$:

$$\underline{[\![\Gamma]\!]}\otimes\underline{[\![\Delta]\!]}\xrightarrow{\varphi(\mathsf{w}_{[\![\Gamma]\!]}\otimes\mathsf{w}_{[\![\Delta]\!]})} 1\xrightarrow{\mathsf{d}'_1}?1\xrightarrow{?\overline{n}}?\mathsf{N}$$

Assume next that $\Gamma,x:\sigma\vdash t:\tau\mid\Delta$ so that we have $[\![t]\!]_{\Gamma,x:\sigma,\Delta}\,\varphi\in\mathcal{L}(\underline{[\![\Gamma]\!]}\otimes\underline{[\![\Delta]\!]}\otimes !(\underline{[\![\sigma]\!]}^\bot),\underline{[\![\tau]\!]}^\bot)$. We set $[\![\lambda x^\sigma\, t]\!]_{\Gamma,\Delta}=\mathsf{cur}([\![t]\!]_{\Gamma,x:\sigma,\Delta}\,\varphi)\in\mathcal{L}(\underline{[\![\Gamma]\!]}\otimes\underline{[\![\Delta]\!]},!(\underline{[\![\sigma]\!]}^\bot)\multimap\underline{[\![\tau]\!]}^\bot)$ and we have $!(\underline{[\![\sigma]\!]}^\bot)\multimap\underline{[\![\tau]\!]}^\bot=(!(\underline{[\![\sigma]\!]}^\bot)\otimes\underline{[\![\tau]\!]})^\bot=\underline{[\![\sigma\Rightarrow\tau]\!]}^\bot$ up to canonical isos.

Assume that $\Gamma\vdash s:\sigma\mid\Delta$ and $\Gamma\vdash t:\tau\mid\Delta$ so that we have $[\![s]\!]_{\Gamma,\Delta}\in\mathcal{L}(\underline{[\![\Gamma]\!]}\otimes\underline{[\![\Delta]\!]},\underline{[\![\sigma]\!]}^\bot)$ and $[\![t]\!]_{\Gamma,\Delta}\in\mathcal{L}(\underline{[\![\Gamma]\!]}\otimes\underline{[\![\Delta]\!]},\underline{[\![\tau]\!]}^\bot)$. So we set $[\![\langle s,t\rangle]\!]_{\Gamma,\Delta}=\langle[\![s]\!]_{\Gamma,\Delta},[\![t]\!]_{\Gamma,\Delta}\rangle\in\mathcal{L}([\![\Gamma]\!]\otimes[\![\Delta]\!],\underline{[\![\sigma]\!]}^\bot\,\&\,\underline{[\![\tau]\!]}^\bot)$ which has the prescribed codomain since $\underline{[\![\sigma]\!]}^\bot\,\&\,\underline{[\![\tau]\!]}^\bot=\underline{[\![\sigma\times\tau]\!]}^\bot$.

Assume that $\Gamma\vdash c\mid\alpha:\sigma,\Delta$ so that we have $[\![c]\!]_{\Gamma,\alpha:\sigma,\Delta}\,\varphi\in\mathcal{L}(\underline{[\![\Gamma]\!]}\otimes\underline{[\![\Delta]\!]}\otimes\underline{[\![\sigma]\!]},\bot)$. Then we set $[\![\mu\alpha^\sigma\, c]\!]_{\Gamma,\Delta}=\mathsf{cur}([\![c]\!]_{\Gamma,\alpha:\sigma,\Delta}\,\varphi)\in\mathcal{L}(\underline{[\![\Gamma]\!]}\otimes\underline{[\![\Delta]\!]},\underline{[\![\sigma]\!]}^\bot)$.

Assume that $\Gamma,x:\sigma\vdash t:\sigma\mid\Delta$ so that we have $[\![t]\!]_{\Gamma,x:\sigma,\Delta}\,\varphi\in\mathcal{L}(\underline{[\![\Gamma]\!]}\otimes\underline{[\![\Delta]\!]}\otimes!(\underline{[\![\sigma]\!]}^\bot),\underline{[\![\sigma]\!]}^\bot)$. We set $[\![\mathsf{fix}\,x^\sigma\, t]\!]_{\Gamma,\Delta}=\overline{\mathsf{fix}}_{\underline{[\![\sigma]\!]}^\bot}\,\mathsf{cur}[\![t]\!]_{\Gamma,x:\sigma,\Delta}\,\varphi\in\mathcal{L}(\underline{[\![\Gamma]\!]}\otimes\underline{[\![\Delta]\!]},\underline{[\![\sigma]\!]}^\bot)$.

Concerning commands, assume that $\Gamma\vdash t:\sigma\mid\Delta$ and that $\Gamma\mid\pi:\sigma\vdash\Delta$ so that we have $[\![t]\!]_{\Gamma,\Delta}\in\mathcal{L}(\underline{[\![\Gamma]\!]}\otimes\underline{[\![\Delta]\!]},\underline{[\![\sigma]\!]}^\bot)$ and $[\![\pi]\!]_{\Gamma,\Delta}\in\mathcal{L}^!([\![\Gamma]\!]\otimes[\![\Delta]\!],[\![\sigma]\!])$ and therefore $[\![\pi]\!]_{\Gamma,\Delta}\in\mathcal{L}(\underline{[\![\Gamma]\!]}\otimes\underline{[\![\Delta]\!]},\underline{[\![\sigma]\!]})$. We define $[\![t*\pi]\!]_{\Gamma,\Delta}$ as the following composition of morphisms in $\mathcal{L}$

$$\underline{[\![\Gamma]\!]}\otimes\underline{[\![\Delta]\!]}\xrightarrow{\mathsf{c}_{[\![\Gamma]\!]\otimes[\![\Delta]\!]}}\underline{[\![\Gamma]\!]}\otimes\underline{[\![\Delta]\!]}\otimes\underline{[\![\Gamma]\!]}\otimes\underline{[\![\Delta]\!]}\xrightarrow{[\![t]\!]\otimes[\![\pi]\!]}\underline{[\![\sigma]\!]}^\bot\otimes\underline{[\![\sigma]\!]}\xrightarrow{\mathsf{ev}}\bot$$

Let us come now to stacks. The morphism $[\![\alpha]\!]_{\Gamma,\alpha:\sigma,\Delta}$ is defined as the following composition of morphisms in $\mathcal{L}^!$

$$[\![\Gamma]\!]\otimes[\![\sigma]\!]\otimes[\![\Delta]\!]\xrightarrow{\mathsf{w}_{[\![\Gamma]\!]}\otimes[\![\sigma]\!]\otimes\mathsf{w}_{[\![\Delta]\!]}}1\otimes[\![\sigma]\!]\otimes 1\xrightarrow{\varphi}[\![\sigma]\!]$$

Remember that we have defined $\overline{\mathsf{succ}}, \overline{\mathsf{pred}} \in \mathcal{L}(\mathsf{N}, \mathsf{N})$, so that we have $!(\overline{\mathsf{succ}}^{\perp}), !(\overline{\mathsf{pred}}^{\perp}) \in \mathcal{L}^!(!(\underline{\mathsf{N}}^{\perp}), !(\underline{\mathsf{N}}^{\perp}))$. Assume that $\Gamma \mid \pi : \iota \vdash \Delta$ so that $[\![\pi]\!]_{\Gamma,\Delta} \in \mathcal{L}^!([\![\Gamma]\!] \otimes [\![\Delta]\!], [\![\iota]\!])$, and we set $[\![\mathsf{succ} \cdot \pi]\!]_{\Gamma,\Delta} = !(\overline{\mathsf{succ}}^{\perp})\, [\![\pi]\!]_{\Gamma,\Delta}$ and $[\![\mathsf{pred} \cdot \pi]\!]_{\Gamma,\Delta} = !(\overline{\mathsf{pred}}^{\perp})\, [\![\pi]\!]_{\Gamma,\Delta}$; both morphisms belong to $\mathcal{L}^!([\![\Gamma]\!] \otimes [\![\Delta]\!], [\![\iota]\!])$.

Remember also that, for any object X of $\mathcal{L}$, we have defined $\overline{\mathsf{if}} \in \mathcal{L}(\mathsf{N} \otimes !X \otimes !X, X)$. Using $*$-autonomy and isos induced by the monoidal structure of $\mathcal{L}$, we can canonically turn this morphism into $\overline{\mathsf{if}}' \in \mathcal{L}(X^{\perp} \otimes !X \otimes !X, \mathsf{N}^{\perp})$. Assume that $X = \underline{P}^{\perp}$ where P is an object of $\mathcal{L}^!$. Then we can set $\overline{\mathsf{If}} = \overline{\mathsf{if}}'^{!} \in \mathcal{L}^!(P \otimes !(\underline{P}^{\perp}) \otimes !(\underline{P}^{\perp}), !(\mathsf{N}^{\perp}))$. Assume that $\Gamma \mid \pi : \sigma \vdash \Delta$ and $\Gamma \vdash t_i : \sigma \mid \Delta$ for $i = 1, 2$. Then we have $[\![\pi]\!]_{\Gamma,\Delta} \in \mathcal{L}^!([\![\Gamma]\!] \otimes [\![\Delta]\!], [\![\sigma]\!])$ and $[\![t_1]\!]^!_{\Gamma,\Delta}, [\![t_2]\!]^!_{\Gamma,\Delta} \in \mathcal{L}^!([\![\Gamma]\!] \otimes [\![\Delta]\!], ![\![\sigma]\!]^{\perp})$, and we define $[\![\mathsf{if}(t_1, t_2, \pi)]\!]_{\Gamma,\Delta}$ as the following composition of morphisms in $\mathcal{L}^!$, using a ternary version of the contraction morphism

$$[\![\Gamma]\!] \otimes [\![\Delta]\!] \xrightarrow{\mathsf{c}^{(3)}_{[\![\Gamma]\!] \otimes [\![\Delta]\!]}} ([\![\Gamma]\!] \otimes [\![\Delta]\!])^{\otimes 3} \xrightarrow{[\![\pi]\!] \otimes [\![t_1]\!]^! \otimes [\![t_2]\!]^!} [\![\sigma]\!] \otimes !([\![\sigma]\!]^{\perp}) \otimes !([\![\sigma]\!]^{\perp}) \xrightarrow{\overline{\mathsf{If}}} !(\mathsf{N}^{\perp})$$

Assume that $\Gamma \vdash \pi : \tau \mid \Delta$ and that $\Gamma \vdash t : \sigma \mid \Delta$ so that $[\![\pi]\!]_{\Gamma,\Delta} \in \mathcal{L}^!([\![\Gamma]\!] \otimes [\![\Delta]\!], [\![\tau]\!])$ and $[\![t]\!]^!_{\Gamma,\Delta} \in \mathcal{L}^!([\![\Gamma]\!] \otimes [\![\Delta]\!], !([\![\sigma]\!]^{\perp}))$, we set $[\![\mathsf{arg}(t) \cdot \pi]\!]_{\Gamma,\Delta} = ([\![t]\!]^!_{\Gamma,\Delta} \otimes [\![\pi]\!]_{\Gamma,\Delta})\, \mathsf{c}_{[\![\Gamma]\!] \otimes [\![\Delta]\!]} \in \mathcal{L}^!([\![\Gamma]\!] \otimes [\![\Delta]\!], [\![\sigma \Rightarrow \tau]\!])$.

Assume last that $\Gamma \mid \pi : \sigma \vdash \Delta$ so that $[\![\pi]\!]_{\Gamma,\Delta} \in \mathcal{L}^!([\![\Gamma]\!] \otimes [\![\Delta]\!], [\![\sigma]\!])$ and we can set $[\![\mathsf{pr}_1 \cdot \pi]\!] = \mathsf{in}^1\, [\![\pi]\!]_{\Gamma,\Delta} \in \mathcal{L}^!([\![\Gamma]\!] \otimes [\![\Delta]\!], [\![\sigma]\!] \oplus [\![\tau]\!])$ and $[\![\mathsf{pr}_2 \cdot \pi]\!]$ is defined similarly.

Assume now that $\mathcal{L}$ is a model of LL with MIX, see Section 3. Here is the interpretation of the MIX constructs of Section 2. If $c = \mathsf{err}$, with $\Gamma \vdash \mathsf{err} \mid \Delta$, then $[\![c]\!]_{\Gamma,\Delta} = \mathsf{mix}^0\, \mathsf{w}_{[\![\Gamma]\!] \otimes [\![\Delta]\!]}$. If $c = c_1 \| c_2$ with $\Gamma \vdash c_i \mid \Delta$ for $i = 1, 2$, we set $[\![c]\!]_{\Gamma,\Delta} = \mathsf{mix}^2\, ([\![c_1]\!] \otimes [\![c_2]\!])\, \mathsf{c}_{[\![\Gamma]\!] \otimes [\![\Delta]\!]}$.

▶ **Theorem 3** (Soundness). *Assume that* $\Gamma \vdash c \mid \Delta$ *and that* $c \to c'$. *Then* $[\![c]\!]_{\Gamma,\Delta} = [\![c']\!]_{\Gamma,\Delta}$.

3.2 A continuation category

We recall briefly the connection between this LL-based approach and the Lafont-Reus-Streicher (LRS) [15] approach of continuation categories, see [11] for more details[2]. Let $\mathcal{P} = \mathcal{L}^!$, we have seen that $\mathcal{P}$ is a cocartesian and cartesian category, with $\oplus$ as coproduct and $\otimes$ as product. As object of responses, we take $\Sigma = !\perp$. Let P and Q be objects of $\mathcal{P}$. Then we have $\mathcal{P}(P \otimes Q, \Sigma) = \mathcal{L}^!(P \otimes Q, !\perp) \simeq \mathcal{L}(\underline{P} \otimes \underline{Q}, \perp)$ because $!\_$ is right adjoint to U. Hence $\mathcal{P}(P \otimes Q, \Sigma) \simeq \mathcal{L}(\underline{P}, \underline{Q}^{\perp}) \simeq \mathcal{L}^!(P, !(\underline{Q}^{\perp}))$ by the same adjunction. So setting $\Sigma^Q = !(\underline{Q}^{\perp})$ we have $\mathcal{P}(P \otimes Q, \Sigma) \simeq \mathcal{P}(P, \Sigma^Q)$. Hence Σ is a baseable object of $\mathcal{P}$.

The category $\Sigma^{\mathcal{P}}$ of *negated objects* has the same objects as $\mathcal{P}$, and $\Sigma^{\mathcal{P}}(P, Q) = \mathcal{P}(\Sigma^P, \Sigma^Q)$. It is a cartesian closed category with product $P \times Q = P \oplus Q$ and object of morphisms $P \Rightarrow Q = \Sigma^P \otimes Q$ as can easily be checked, using the fact that Σ is baseable. In the LRS setting, interpretation of types is done in $\mathcal{P}$, setting $[\![\sigma \Rightarrow \tau]\!] = \Sigma^{[\![\sigma]\!]} \otimes [\![\tau]\!]$ and, given contexts $\Gamma = (x_1 : \sigma_1, \dots, x_n : \sigma_n)$ and $\Delta = (\alpha_1 : \tau_1, \dots, \alpha_k : \tau_k)$, a term t such that $\Gamma \vdash t : \sigma \mid \Delta$ is interpreted as $[\![t]\!]_{\Gamma,\Delta} \in \mathcal{P}(\Sigma^{[\![\sigma_1]\!]} \times \cdots \times \Sigma^{[\![\sigma_n]\!]} \times [\![\tau_1]\!] \times \cdots \times [\![\tau_k]\!], \Sigma^{[\![\sigma]\!]})$, a command c such that $\Gamma \vdash c \mid \Delta$ is interpreted as $[\![c]\!]_{\Gamma,\Delta} \in \mathcal{P}(\Sigma^{[\![\sigma_1]\!]} \times \cdots \times \Sigma^{[\![\sigma_n]\!]} \times [\![\tau_1]\!] \times \cdots \times [\![\tau_k]\!], \Sigma)$ and a stack π such that $\Gamma \mid \pi : \tau \vdash \Delta$ is interpreted as $[\![\pi]\!]_{\Gamma,\Delta} \in \mathcal{P}(\Sigma^{[\![\sigma_1]\!]} \times \cdots \times \Sigma^{[\![\sigma_n]\!]} \times [\![\tau_1]\!] \times \cdots \times [\![\tau_k]\!], [\![\tau]\!])$ and it is easily checked again that this interpretation is exactly the same as the one described above, up to the identification of $\mathcal{P}(P, !X)$ with $\mathcal{L}(\underline{P}, X)$.

[2] This paper establishes the correspondence with Selinger control categories [14] which are equivalent to continuation categories. They use therefore a negative translation whereas we use a positive one.

4 Relational semantics

In this most simple and canonical interpretation of LL, $\mathcal{L}$ is the category **Rel** whose objects are sets[3] and where $\mathbf{Rel}(X,Y) = \mathcal{P}(X \times Y)$, composition being defined as the usual composition of relations. We recall that the tensor unit is $1 = \{*\}$ (arbitrary one-point set), that $X \otimes Y = X \times Y$ with tensor product of morphisms defined accordingly, that $X \multimap Y = X \times Y$ (and evaluation defined in the obvious way), that $\bot = 1$ so that $X^\bot = X$ up to canonical iso. This category is countably cartesian, with cartesian product $\&_{i\in I} X_i = \bigcup_{i\in I}(\{i\} \times X_i)$ (disjoint union) and projections defined in the obvious way ($\mathsf{pr}^i = \{((i,a),a) \mid a \in X_i\}$). It is cocartesian with coproducts defined exactly as products and injections given by $\mathsf{in}^i = \{(a,(i,a)) \mid a \in X_i\}$. It has an exponential functor defined on objects by $!X = \mathcal{M}_{\mathrm{fin}}(X)$, the set of all finite multisets[4] of elements of X. On morphisms, this functor is defined by $!f = \{([a_1,\dots,a_n],[b_1,\dots,b_n]) \mid (a_i,b_i) \in f \text{ for each } i\}$. Dereliction (counit) is given by $\mathsf{der}_X = \{([a],a) \mid a \in X\}$ and digging (comultiplication) is given by $\mathsf{dig}_X = \{(m_1+\dots+m_k,[m_1,\dots,m_k]) \mid m_1,\dots,m_k \in !X\}$. The symmetric monoidality isos are given by $\mathsf{m}^{(0)} = \{(*,[])\}$ and

$$\mathsf{m}^{(2)}_{X,Y} = \{(([a_1,\dots,a_n],[b_1,\dots,b_k]),[(1,a_1),\dots,(1,a_n),(2,b_1),\dots,(2,b_k)]) \mid a_1,\dots,a_n \in X \text{ and } b_1,\dots,b_k \in Y\}$$

Let $P = (\underline{P},\mathsf{h}_P)$ be an object of $\mathbf{Rel}^!$ and X be an object of **Rel**. Given $f \in \mathbf{Rel}(\underline{P},X)$, the generalized promotion $f^! \in \mathbf{Rel}^!(P,!X)$ is given by $f^! = \{(b,[a_1,\dots,a_n]) \mid \exists b_1,\dots,b_n \in \underline{P}\ (b,[b_1,\dots,b_n]) \in \mathsf{h}_P \text{ and } (b_i,a_i) \in f \text{ for each } i\}$. The n-ary contraction $\mathsf{c}^{(n)}_P \in \mathbf{Rel}^!(P,P^{\otimes n})$ is given by $\mathsf{c}^{(n)}_P = \{(a,(a_1,\dots,a_n)) \mid (a,[a_1,\dots,a_n]) \in \mathsf{h}_P\}$. In particular (0-ary case) we have $\mathsf{w}_P = \{(a,*) \mid (a,[]) \in \mathsf{h}_P\}$. The next easy lemma is essential for computing the interpretation of expressions, using *eg.* the formalism of Section 4.1.

▶ **Lemma 4.** *Let P_1 and P_2 be objects of $\mathbf{Rel}^!$. One has $((a,b),[(a_1,b_1),\dots,(a_n,b_n)]) \in \mathsf{h}_{P_1\otimes P_2}$ iff $(a,[a_1,\dots,a_n]) \in \mathsf{h}_{P_1}$ and $(b,[b_1,\dots,b_n]) \in \mathsf{h}_{P_2}$. And, given $l \in \{1,2\}$, one has $((l,a),[b_1,\dots,b_n]) \in \mathsf{h}_{P_1\oplus P_2}$ iff, for each $i = 1,\dots,n$, one has $b_i = (l,a_i)$ for some a_i, and moreover $(a,[a_1,\dots,a_n]) \in \mathsf{h}_{P_l}$.*

For each set X, we can define a fix-point operator as a least fix-point wrt. morphism inclusion as $\overline{\mathsf{fix}}_X = \{(m_1+\dots+m_k+[([a_1,\dots,a_k],a)],a) \mid \forall i\ (m_i,a_i) \in \overline{\mathsf{fix}}_X\}$.

The object of natural numbers is the set $\mathbb{N}$, the morphisms $\overline{\mathsf{succ}}$ and $\overline{\mathsf{pred}}$ are given by $\overline{\mathsf{succ}} = \{(n,n+1) \mid n \in \mathbb{N}\}$, $\overline{\mathsf{pred}} = \{(0,0)\cup\{(n+1,n) \mid n \in \mathbb{N}\}$. When $X = \underline{P}^\bot$ where P is an object of $\mathbf{Rel}^!$, the corresponding coalgebra morphism $\overline{\mathsf{if}}_X \in \mathbf{Rel}^!(P\otimes!\underline{P}^\bot\otimes!\underline{P}^\bot,!\mathsf{N}^\bot)$ is given by $\overline{\mathsf{if}}_X = \{(a,[a_1,\dots,a_k],[a_{k+1},\dots,a_l],[n_1,\dots,n_l]) \mid n_1 = \dots = n_k = 0 \text{ and } n_{k+1},\dots,n_l \neq 0 \text{ and } (a,[a_1,\dots,a_l]) \in \mathsf{h}_P\}$. This model of LL is also a model of MIX. It suffices to take $\mathsf{mix}^0 = \{(*,*)\}$ and $\mathsf{mix}^2 = \{((*,*),*))\}$ and these morphisms define clearly a structure of commutative $\otimes$-monoid on $\bot$.

So a typing variable context $\Gamma = (x_1 : \sigma_1,\dots,x_n : \sigma_n)$ is interpreted as a set of tuples $(m_1,\dots,m_n)$ where $m_i \in \mathcal{M}_{\mathrm{fin}}([\![\sigma_i]\!])$ for each i, a typing name context $\Delta = (\alpha_1 : \tau_1,\dots,\alpha_k : \tau_k)$ is interpreted as a set of tuples $(a_1,\dots,a_k)$ where $a_j \in [\![\tau_j]\!]$ for each j.

[3] All sets can be assumed to be at most countable, this is a very reasonable assumption which is preserved by all the constructions that we introduce.

[4] We use $[a_1,\dots,a_k]$ for the multiset whose elements are $a_1,\dots,a_n$, taking multiplicities into account and we use $m+m'$ for the disjoint union of the multiset m and m' which is a natural notation since multisets are $\mathbb{N}$-valued functions. Similarly if $k \in \mathbb{N}$ and m is a multiset, $km = m+\dots+m$ (k times).

With these notations for Γ and Δ, if $\Gamma \vdash M : \tau \mid \Delta$ then $[\![M]\!]_{\Gamma,\Delta}$ is a set of tuples $(m_1, \dots, m_n, a_1, \dots, a_k, a)$ where $a \in [\![\tau]\!]$, if $\Gamma \mid \pi : \sigma \vdash \Delta$ then $[\![\pi]\!]_{\Gamma,\Delta}$ is a set of tuples $(m_1, \dots, m_n, a_1, \dots, a_k, a)$ where $a \in [\![\sigma]\!]$, and if $\Gamma \vdash c \mid \Delta$ then $[\![c]\!]_{\Gamma,\Delta}$ is a set of tuples $(m_1, \dots, m_n, a_1, \dots, a_k)$. In all cases $m_i \in \mathcal{M}_{\mathrm{fin}}([\![\sigma_i]\!])$ and $a_j \in [\![\tau_j]\!]$ for each i and j.

4.1 Interpretation as a type deduction system

We introduce a typing system extending the one of [16] to represent the relational denotational semantics described above. A *semantic variable context* is a sequence $\Phi = (x_1 : m_1 : \sigma_1, \dots, x_n : m_n : \sigma_n)$ where $m_i \in !\underline{[\![\sigma_i]\!]}^{\perp}$ for each i and variables are pairwise distinct. A *semantic name context* is a sequence $\Psi = (\alpha_1 : a_1 : \tau_1, \dots, \alpha_k : a_k : \tau_k)$ where $a_i \in \underline{[\![\tau_i]\!]}$ for each i and the names are pairwise distinct. We also define the underlying typing contexts $\mathsf{u}(\Phi) = (x_1 : \sigma_1, \dots, x_n : \sigma_n)$ and $\mathsf{u}(\Psi) = (\alpha_1 : \tau_1, \dots, \alpha_k : \tau_k)$ as well as the underlying tuples $\langle\Phi\rangle = (m_1, \dots, m_n)$ and $\langle\Psi\rangle = (a_1, \dots, a_k)$. We extend multiset addition to tuples of multisets componentwise.

One has $\langle\Phi\rangle \in \underline{[\![\mathsf{u}(\Phi)]\!]}$ and similarly for Ψ. Given a variable context $\Gamma = (x_1 : \sigma_1, \dots, x_n : \sigma_n)$ one defines the corresponding *zero semantic context* $0_\Gamma = (x_1 : [] : \sigma_1, \dots, x_n : [] : \sigma_n)$.

We define now this typing system. Its main property (proven by an easy induction on expressions) is that $\Phi \vdash t : a : \sigma \mid \Psi$ iff $(\langle\Phi\rangle, \langle\Psi\rangle, a) \in [\![t]\!]_{\mathsf{u}(\Phi),\mathsf{u}(\Psi)}$ and $\Phi \mid \pi : a : \sigma \vdash \Psi$ iff $(\langle\Phi\rangle, \langle\Psi\rangle, a) \in [\![\pi]\!]_{\mathsf{u}(\Phi),\mathsf{u}(\Psi)}$, and also $\Phi \vdash c \mid \Psi$ iff $(\langle\Phi\rangle, \langle\Psi\rangle) \in [\![c]\!]_{\mathsf{u}(\Phi),\mathsf{u}(\Psi)}$. Here are the axioms and deduction rules:

$$\frac{}{0_\Gamma, x : [a] : \sigma \vdash x : a : \sigma \mid \Psi} \qquad \frac{}{0_\Gamma \mid \alpha : a : \sigma \vdash \alpha : a : \sigma, \Psi}$$

if $(\langle\Psi\rangle, 0_{[\![\mathsf{u}(\Psi)]\!]}) \in \mathsf{h}_{[\![\mathsf{u}(\Psi)]\!]}$.

$$\frac{\Phi_1 \vdash t : a : \sigma \mid \Psi_1 \qquad \Phi_2 \mid \pi : a : \sigma \vdash \Psi_2}{\Phi \vdash t * \pi \mid \Psi}$$

if $\mathsf{u}(\Phi_i) = \mathsf{u}(\Phi)$ and $\mathsf{u}(\Psi_i) = \mathsf{u}(\Psi)$ for $i = 1, 2$, $\langle\Phi\rangle = \langle\Phi_1\rangle + \langle\Phi_2\rangle$ and $(\langle\Psi\rangle, [\langle\Psi_1\rangle, \langle\Psi_2\rangle]) \in \mathsf{h}_{[\![\Delta]\!]}$.

$$\frac{\Phi, x : m : \sigma \vdash t : b : \tau \mid \Psi}{\Phi \vdash \lambda x^\sigma\, t : (m, b) : \sigma \Rightarrow \tau \mid \Psi} \qquad \frac{\Phi \vdash s : a : \sigma \mid \Psi \qquad \mathsf{u}(\Phi) \vdash t : \tau \mid \mathsf{u}(\Psi)}{\Phi \vdash \langle s, t\rangle : (1, a) : \sigma \times \tau \mid \Psi}$$

$$\frac{\mathsf{u}(\Phi) \vdash s : \sigma \mid \mathsf{u}(\Psi) \qquad \Phi \vdash t : b : \sigma \mid \Psi}{\Phi \vdash \langle s, t\rangle : (2, b) : \sigma \times \tau \mid \Psi} \qquad \frac{\Phi_0 \mid \pi : b : \tau \vdash \Psi_0 \qquad (\Phi_i \vdash t : a_i : \sigma \mid \Psi_i)_{i=1}^k}{\Phi \mid \mathsf{arg}(t) \cdot \pi : ([a_1, \dots, a_k], b) : \sigma \Rightarrow \tau \vdash \Psi}$$

$$\frac{\Phi_0, x : [a_1, \dots, a_k] : \sigma \vdash t : a : \sigma \mid \Psi_0 \qquad (\Phi_i \vdash \mathsf{fix}\, x^\sigma\, t : a_i : \sigma \mid \Psi_i)_{i=1}^k}{\Phi \vdash \mathsf{fix}\, x^\sigma\, t : a : \sigma \mid \Psi}$$

if $\mathsf{u}(\Phi_i) = \mathsf{u}(\Phi)$, $\mathsf{u}(\Psi_i) = \mathsf{u}(\Psi)$ for each $i = 0, \dots, k$, $\langle\Phi\rangle = \langle\Phi_0\rangle + \cdots + \langle\Phi_k\rangle$ and $(\langle\Psi\rangle, [\langle\Psi_0\rangle, \dots, \langle\Psi_k\rangle]) \in \mathsf{h}_{[\![\mathsf{u}(\Psi)]\!]}$, for the two last deduction rules.

$$\frac{\Phi \mid \pi : a : \sigma \vdash \Psi}{\Phi \mid \mathsf{pr}_1 \cdot \pi : (1, a) : \sigma \times \tau \vdash \Psi} \qquad \frac{\Phi \mid \pi : b : \tau \vdash \Psi}{\Phi \mid \mathsf{pr}_2 \cdot \pi : (2, b) : \sigma \times \tau \vdash \Psi}$$

$$\frac{\Phi \vdash c \mid \alpha : a : \sigma, \Psi}{\Phi \vdash \mu\alpha^\sigma\, c : a : \sigma \mid \Psi} \qquad \frac{(\langle\Psi\rangle, 0_{[\![\mathsf{u}(\Psi)]\!]}) \in \mathsf{h}_{[\![\mathsf{u}(\Psi)]\!]} \qquad n \in \mathbb{N}}{0_\Gamma \vdash \underline{n} : n : \iota \mid \Psi}$$

$$\frac{\Phi \mid \pi : [n_1 + 1, \dots, n_k + 1] : \iota \vdash \Psi}{\Phi \mid \mathsf{succ} \cdot \pi : [n_1, \dots, n_k] : \iota \vdash \Psi} \qquad \frac{\Phi \mid \pi : k[0] + [n_1, \dots, n_l] : \iota \vdash \Psi}{\Phi \mid \mathsf{pred} \cdot \pi : k[0] + [n_1 + 1, \dots, n_l + 1] : \iota \vdash \Psi}$$

$$\frac{\Phi_0 \mid \pi : a : \sigma \vdash \Psi_0 \qquad (\Phi_i \vdash t_1 : a_i : \sigma \mid \Psi_i)_{i=1}^{k} \qquad (\Phi_i \vdash t_2 : a_i : \sigma \mid \Psi_i)_{i=k+1}^{l}}{\Phi \mid \mathsf{if}(t_1, t_2) \cdot \pi : \iota : k[0] + [n_1 + 1, \dots, n_{l-k} + 1] \vdash \Psi}$$

if $(a, [a_1, \dots, a_l]) \in \mathsf{h}_{[\![\sigma]\!]}$, $\mathsf{u}(\Phi_i) = \mathsf{u}(\Phi)$, $\mathsf{u}(\Psi_i) = \mathsf{u}(\Psi)$ for each i, $\langle\Phi\rangle = \langle\Phi_0\rangle + \dots + \langle\Phi_l\rangle$ and $(\langle\Psi\rangle, [\langle\Psi_0\rangle, \dots, \langle\Psi_l\rangle]) \in \mathsf{h}_{[\![\mathsf{u}(\Psi)]\!]}$. For classical PCF with MIX, we add the rules:

$$\frac{(\langle\Psi\rangle, 0_{[\![\mathsf{u}(\Psi)]\!]}) \in \mathsf{h}_{[\![\mathsf{u}(\Psi)]\!]}}{0_\Gamma \vdash \mathsf{err} \mid \Psi} \qquad \frac{\Phi_1 \vdash c_1 \mid \Psi_1 \qquad \Phi_2 \vdash c_2 \mid \Psi_2}{\Phi \vdash c_1 \| c_2 \mid \Psi}$$

if $\mathsf{u}(\Phi_i) = \mathsf{u}(\Phi)$, $\mathsf{u}(\Psi_i) = \mathsf{u}(\Psi)$ for $i = 1, 2$, $\langle\Phi\rangle = \langle\Phi_1\rangle + \langle\Phi_2\rangle$ and $(\langle\Psi\rangle, [\langle\Psi_1\rangle, \langle\Psi_2\rangle]) \in \mathsf{h}_{[\![\mathsf{u}(\Psi)]\!]}$.

5 Adequacy

Our goal here is to prove that, in the full calculus (including the MIX constructions), if an almost closed command has a non-empty relational semantics, then its $\to$-reduction terminates. In other words, an almost closed command typable in the semantic typing system is $\to$-normalizing. Let $\mathcal{N}$ be the set of all $\to$-normalizing almost closed commands.

Let us say that a term t (resp. a stack π) is almost closed of type σ if $\vdash t : \sigma \mid \Delta$ (resp. $\mid \pi : \sigma \vdash \Delta$) for some ground name context Δ (that is, for any ground name context where all free names appear). Observe that if t and π are an almost closed term and an almost closed stack of the same type, then $t * \pi$ is an almost closed command.

By induction on σ, we define, for each $a \in [\![\sigma]\!]$, a set $\|a\|^\sigma$ of almost closed stacks of type σ. We use the notation $|a|^\sigma$ for the set of all almost closed terms t of type σ such that $t * \pi \in \mathcal{N}$ for all $\pi \in \|a\|^\sigma$. Given $a_1, \dots, a_n \in [\![\sigma]\!]$, we set $|[a_1, \dots, a_n]|^\sigma = \bigcap_{i=1}^n |a_i|^\sigma$.

The most important part of the definition is the base case: given $m = [n_1, \dots, n_k] \in [\![\iota]\!] = !(\mathsf{N}^\perp)$, we define $\|m\|^\iota$ as the set of all almost closed stacks π of type ι such that $\forall i \in \{1, \dots, k\}$ $\underline{n_i} * \pi \in \mathcal{N}$. This set contains all names (considered of type ι) and hence is never empty.

The inductive step follows the general pattern of classical reducibility. Let σ and τ be types, let $a_1, \dots, a_n \in [\![\sigma]\!]$ and $b \in [\![\tau]\!]$. We set

$$\|([a_1, \dots, a_n], b)\|^{\sigma \Rightarrow \tau} = \left\{ \mathsf{arg}(t) \cdot \pi \mid t \in |[a_1, \dots, a_n]|^\sigma \text{ and } \pi \in \|b\|^\tau \right\}$$

Let $a \in [\![\sigma]\!]$, we set $\|(1, a)\|^{\sigma \times \tau} = \{\mathsf{pr}_1 \cdot \pi \mid \pi \in \|a\|^\sigma\}$ and $\|(2, b)\|^{\sigma \times \tau}$ is defined similarly for $b \in [\![\tau]\!]$.

▶ **Theorem 5** (Adequacy). *Let $\Phi = (x_1 : m_1 : \sigma_1, \dots, x_n : m_n : \sigma_n)$ and $\Psi = (\alpha_1 : a_1 : \tau_1, \dots, \alpha_k : a_k : \tau_k)$ be semantic contexts. Let σ be a type, t be a term, c be a command and π be a stack such that*

- $\Phi \vdash t : a : \sigma \mid \Psi$
- *resp.* $\Phi \vdash c \mid \Psi$,
- *resp.* $\Phi \mid \pi : a : \sigma \vdash \Psi$.

Then, for all $t_1 \in |m_1|^{\sigma_1}, \dots, t_n \in |m_n|^{\sigma_n}$ and all $\pi_1 \in \|a_1\|^{\tau_1}, \dots, \pi_k \in \|a_k\|^{\tau_k}$, one has

- $t\,[t_1/x_1, \dots, t_n/x_n]\,[\pi_1/\alpha_1, \dots, \pi_k/\alpha_k] \in |a|^\sigma$
- *resp.* $c\,[t_1/x_1, \dots, t_n/x_n]\,[\pi_1/\alpha_1, \dots, \pi_k/\alpha_k] \in \mathcal{N}$,
- *resp.* $\pi\,[t_1/x_1, \dots, t_n/x_n]\,[\pi_1/\alpha_1, \dots, \pi_k/\alpha_k] \in \|a\|^\sigma$.

So, if an almost closed command c has a non-empty interpretation, it normalizes for the $\to$-reduction to a uniquely defined normal almost closed command which can easily be retrieved from the semantics of c. For instance if c satisfies $\vdash c \mid \Delta$ where $\Delta = (\alpha_1 : \iota, \alpha_2 : \iota, \alpha_3 : \iota)$,

and if we have $\vdash c \mid \alpha_1 : [0,7] : \iota, \alpha_2 : [3] : \iota, \alpha_3 : [] : \iota$, then c normalizes by Theorem 5. Its normal form c_0 satisfies $\vdash c_0 \mid \alpha_1 : \iota, \alpha_2 : \iota, \alpha_3 : \iota$ by Proposition 1 and hence must be of shape $(n_1^1 * \alpha_1) \| \cdots \| (n_1^{l_1} * \alpha_1) \| (n_2^1 * \alpha_2) \| \cdots \| (n_2^{l_2} * \alpha_2) \| (n_3^1 * \alpha_3) \| \cdots \| (n_3^{l_3} * \alpha_3)$ up to associativity and commutativity of $\|$, by the final considerations of Section 2. By definition of the interpretation, $[\![c_0]\!]_{(),\Delta} = \{([n_1^1, \ldots, n_1^{l_1}], [n_2^1, \ldots, n_2^{l_2}], [n_3^1, \ldots, n_3^{l_3}])\}$. But by Theorem 3 we have $[\![c]\!]_{(),\Delta} = [\![c_0]\!]_{(),\Delta}$ and hence *we must have* $l_1 = 2$, $l_2 = 1$, $l_3 = 0$, $n_1^1 = 0$, $n_1^2 = 7$ (or conversely) and $n_1^2 = 3$. This adequacy property entails that denotational equivalence of terms implies their observational equivalence (to be suitably defined)[5].

The considerations above show that the interpretation of an almost closed command contains at most one element. This can also be proved purely semantically, endowing the relational semantics with a binary *coherence relation*.

If c does not contain MIX constructs, we know that it will reduce to a normal command of shape $\underline{n} * \alpha$, but the model does not reflect this property that we proved syntactically in Section 2. We introduce now a light refinement of the relational model which takes this *uniqueness of values* property into account, and therefore rejects the MIX constructs.

6 A semantic account of uniqueness of values

This model originates from the observation made independently by several authors[6] at an early stage of the development of LL that, in a multiplicative proof-net, there is a simple relation between the number of $\otimes$'s and of $\parr$'s.

A *weighted set* is a pair $X = (|X|, \gamma_X)$ where $|X|$ is a set and $\gamma_X : |X| \to \mathbb{Z}$ is a function. If we think of a as a proof tree in (constant-free) multiplicative LL (MLL) with only one conclusion (the root of the tree), then $\gamma_X(a) = p - t$ where p is the number of $\parr$ and t is the number of $\otimes$ binary connectives occurring in a. If such a multiplicative proof tree can be sequentialized into a sequent calculus proof in MLL, then $p - t = 1$, see *eg.* [6], pages 250-251 (the converse is not true). This intuition explains the next definitions. One sets $\mathsf{C}(X) = \{x \subseteq |X| \mid \forall a \in x \ \gamma_X(a) = 1\}$.

Let **RelW** be the category of weighted sets and such that $\mathbf{RelW}(X,Y) = \{t \subseteq |X| \times |Y| \mid \forall (a,b) \in t \ \gamma_X(a) = \gamma_Y(b)\}$. Then $\mathsf{Id}_X = \{(a,a) \mid a \in |X|\} \in \mathbf{RelW}(X,X)$ and the relational composition of two morphisms is a morphism, so **RelW** is a category.

One defines the weighted set 1 by $|1| = \{*\}$ (a singleton) and $\gamma_1(*) = 1$. Given two weighted sets X_1 and X_2, one defines $X_1 \otimes X_2$ by $|X_1 \otimes X_2| = |X_1| \times |X_2|$ and $\gamma_{X_1 \otimes X_2}(a_1, a_2) = \gamma_{X_1}(a_1) + \gamma_{X_2}(a_2) - 1$. Given $t_i \in \mathbf{RelW}(X_i, Y_i)$ for $i = 1, 2$, one defines $t_1 \otimes t_2$ as in **Rel**, then it is clear that $t_1 \otimes t_2 \in \mathbf{RelW}(X_1 \otimes X_2, Y_1 \otimes Y_2)$ and that this operation is a functor. Moreover, the usual bijections $|1 \otimes X| \to |X|$, $|X \otimes 1| \to |X|$ and $|(X_1 \otimes X_2) \otimes X_3| \to |X_1 \otimes (X_2 \otimes X_3)|$ are isos in **RelW**. Indeed we have $\gamma_{1 \otimes X}(*, a) = 1 + \gamma_X(a) - 1 = \gamma_X(a)$ and $\gamma_{(X_1 \otimes X_2) \otimes X_3}((a_1, a_2), a_3) = \gamma_{X_1}(a_1) + \gamma_{X_2}(a_2) + \gamma_{X_3}(a_3) - 2 = \gamma_{X_1 \otimes (X_2 \otimes X_3)}(a_1, (a_2, a_3))$.

In that way, we have equipped **RelW** with a structure of symmetric monoidal category. We check that it is closed. Given two weighted sets X and Y, let $X \multimap Y = (|X| \times |Y|, \gamma_{X \multimap Y})$ and $\gamma_{X \multimap Y}(a,b) = \gamma_Y(b) - \gamma_X(a) + 1$. Observe that $\mathsf{C}(X \multimap Y) = \mathbf{RelW}(X,Y)$.

Then $\mathsf{ev} = \{(((a,b),a),b) \mid a \in |X| \text{ and } b \in |Y|\}$ belongs to $\mathbf{RelW}((X \multimap Y) \otimes X, Y)$. Indeed we have $\gamma_{(X \multimap Y) \otimes X}((a,b),a) = \gamma_Y(b) - \gamma_X(a) + 1 + \gamma_X(a) - 1 = \gamma_Y(b)$.

[5] The converse implication (full abstraction) is far from being true.
[6] At least: Girard, Danos and Regnier, Métayer, Fleury and Rétoré, Guerrini...

Let Z be another weighted set and let $((c,a),b) \in |(Z \otimes X) \multimap Y|$. Then we have $\gamma_{(Z\otimes X)\multimap Y}((c,a),b) = \gamma_Y(b) - (\gamma_Z(c)+\gamma_X(a)-1)+1 = \gamma_Y(b)-\gamma_Z(c)-\gamma_X(a)+2$. On the other hand we have $\gamma_{Z\multimap(X\multimap Y)}(c,(a,b)) = (\gamma_Y(b)-\gamma_X(a)+1)-\gamma_Z(c)+1 = \gamma_Y(b)-\gamma_Z(c)-\gamma_X(a)+2$ and therefore, given $t \in \mathbf{RelW}(Z \otimes X, Y)$, we have $\mathsf{cur}(t) = \{(c,(a,b)) \mid ((c,a),b) \in t\} \in \mathbf{RelW}(Z, X \multimap Y)$. This shows that **RelW** is closed.

Let $\bot = (\{*\}, \gamma_\bot)$ with $\gamma_\bot(*) = -1$. Then we have $\gamma_{X\multimap\bot}(a,*) = -1 - \gamma_X(a) + 1 = -\gamma_X(a)$. It follows that the canonical morphism $\eta_X \in \mathbf{RelW}(X, (X \multimap \bot) \multimap \bot)$ given by $\eta_X = \mathsf{cur}(\mathsf{ev}\ \sigma_{X,X\multimap\bot})$ (where σ is the symmetry natural iso associated with the symmetric monoidal closed structure of **RelW**) is an iso in **RelW**. This shows that, equipped with $\bot$ as dualizing object, the symmetric monoidal closed category **RelW** is *-autonomous.

The co-tensor product, called *par*, is the operation defined by $X \parr Y = (X^\bot \otimes Y^\bot)^\bot$ and is characterized by $|X \parr Y| = |X| \times |Y|$ and $\gamma_{X\parr Y}(a,b) = \gamma_X(a) + \gamma_Y(b) + 1$.

Let $X^\bot = (|X|, -\gamma_X)$. Then $X^\bot$ is naturally isomorphic to $X \multimap \bot$ and defines a strictly involutive functor $\mathbf{RelW} \to \mathbf{RelW}^{\mathsf{op}}$. Its action on morphisms is contraposition: $t^\bot = \{(b,a) \mid (a,b) \in t\} \in \mathbf{RelW}(Y^\bot, X^\bot)$ for any $t \in \mathbf{RelW}(X,Y)$.

The category **RelW** is cartesian and cocartesian. Given a family $(X_i)_{i\in I}$ of objects, let $X = \&_{i\in I} X_i$ be defined by $|X| = \bigcup_{i\in I}\{i\} \times |X_i|$ and $\gamma_X(i,a) = \gamma_{X_i}(a)$. Let $\mathsf{pr}^i = \{((i,a),a) \mid a \in |X_i|\} \in \mathbf{RelW}(X, X_i)$. Then $(X, (\mathsf{pr}^i)_{i\in I})$ is a cartesian product of the family $(X_i)_{i\in I}$. The coproduct is defined in a completely similar way. Observe that the product of the empty family (the terminal object) is $\top = (\emptyset, \emptyset)$, which is also the initial object of **RelW**.

Let $!X = (\mathcal{M}_{\mathrm{fin}}(|X|), \gamma_{!X})$ where

$$\gamma_{!X}([a_1,\dots,a_n]) = \gamma_{X^{\otimes n}}(a_1,\dots,a_n) = -n+1+\sum_{i=1}^n \gamma_X(a_i) = 1+\sum_{i=1}^n(\gamma_X(a_i)-1)\,.$$

Given $t \in \mathbf{RelW}(X,Y)$, it is clear that $!t \in \mathbf{RelW}(!X, !Y)$ where $!t$ is defined as in **Rel**.

So $!\_$ is a functor $\mathbf{RelW} \to \mathbf{RelW}$. We equip this functor with a structure of comonad. For each object X, let $\mathsf{der}_X = \{([a],a) \mid a \in |X|\}$. Since $\gamma_{!X}([a]) = \gamma_X(a)$, we have $\mathsf{der}_X \in \mathbf{RelW}(!X, X)$. The naturality of der_X is obvious (it already holds in **Rel**).

One defines also $\mathsf{dig}_X = \{(m_1 + \dots + m_k, [m_i,\dots,m_k]) \mid k \in \mathbb{N} \text{ and } \forall i\, m_i \in \mathcal{M}_{\mathrm{fin}}(|X|)\}$. Let $m_1,\dots,m_k \in \mathcal{M}_{\mathrm{fin}}(|X|)$, and let us write $m_i = [a^i_1,\dots,a^i_{k_i}]$. We have

$$\begin{aligned}\gamma_{!!X}([m_1,\dots,m_k]) &= 1+\sum_{i=1}^k(\gamma_{!X}(m_i)-1) = 1+\sum_{i=1}^k(1+\big(\sum_{j=1}^{k_i}\gamma_X(a^i_j)-1\big)-1)\\ &= \gamma_{!X}(m_1+\dots+m_k)\end{aligned}$$

and therefore $\mathsf{dig}_X \in \mathbf{RelW}(!X, !!X)$. One proves easily that $(!X, \mathsf{der}_X, \mathsf{dig}_X)$ defines a comonad (the definition of this structure is the same as in **Rel**).

Last we check that the standard Seely isos of **Rel** are morphisms in **RelW**. The 0-ary iso is $\mathsf{m}^{(0)} = \{(*,[])\}$ and belongs to $\mathbf{RelW}(1, !\top)$ since $\gamma_{!\top}([]) = 1$. The binary version is $\mathsf{m}^{(2)}_{X,Y} = \{(([a_1,\dots,a_n],[b_1,\dots,b_p]),\ [(1,a_1),\dots,(1,a_n),(2,b_1),\dots,(2,b_p)]) \mid \forall i\, a_i \in |X| \text{ and } \forall j\, b_j \in |Y|\}$ and we prove that $\mathsf{m}^{(2)}_{X,Y} \in \mathbf{RelW}(!X \otimes !Y, !(X \,\&\, Y))$: indeed, with the notations of this definition, we have

$$\gamma_{!X\otimes!Y}([a_1,\dots,a_n],[b_1,\dots,b_p]) = 1+\sum_{i=1}^n(\gamma_X(a_i)-1)+1+\sum_{j=1}^p(\gamma_Y(b_j)-1)-1$$

$$= 1+\sum_{i=1}^n(\gamma_X(a_i)-1)+\sum_{j=1}^p(\gamma_Y(b_j)-1) = \gamma_{!(X\&Y)}([(1,a_1),\dots,(1,a_n),(2,b_1),\dots,(2,b_p)]))$$

This ends the description of the purely logical structures of the model.

The object N of natural numbers (in the sense of Section 3) is the coproduct of ω copies of 1, so $|\mathsf{N}| = \mathbb{N}$ and $\gamma_{\mathsf{N}}(n) = 1$ for each $n \in \mathbb{N}$. The morphisms $\overline{\mathsf{succ}}$, $\overline{\mathsf{pred}}$ and $\overline{\mathsf{if}}$ as defined in Section 4 in the category **Rel** are also morphisms in **RelW** simply because they are defined using the universal property of N. Last, for any object X of **RelW**, the fix-point operator $\overline{\mathsf{fix}}_{|X|}$ as defined in Section 4 is also a morphism $!(!X \multimap X) \to X$ in **RelW**.

So **RelW** is a model of classical PCF in the sense of Section 3.1. Let $[\![\sigma]\!]^{\mathsf{w}}$ be the interpretation of the type σ in the category $\mathbf{RelW}^!$, we have $|\underline{[\![\sigma]\!]^{\mathsf{w}}}| = \underline{[\![\sigma]\!]}$ and $\mathsf{h}_{[\![\sigma]\!]^{\mathsf{w}}} = \mathsf{h}_{[\![\sigma]\!]}$ (as relations). If e is an expression typable in contexts Γ, Δ, we denote with $[\![e]\!]^{\mathsf{w}}_{\Gamma,\Delta}$ the interpretation of e in **RelW** (if e is a command or a term) or in $\mathbf{RelW}^!$ (if e is a stack).

▶ **Proposition 6.** *For any expression of classical PCF e typable in contexts Γ and Δ, one has $[\![e]\!]^{\mathsf{w}}_{\Gamma,\Delta} = [\![e]\!]_{\Gamma,\Delta}$ (as relations).*

This is due to the fact that the basic LL constructs are interpreted by the same relations in both models.

Rejection of MIX and uniqueness of values. Observe that $\mathsf{mix}^0 = \{(*,*)\} \in \mathbf{Rel}(1,\bot)$ and $\mathsf{mix}^2 = \{((*,*),*)\} \in \mathbf{Rel}(\bot \otimes \bot, \bot)$ *are not* morphisms in **RelW**: for mix^0, this is due to the fact that $\gamma_1(*) = 1$ and $\gamma_\bot(*) = -1$ and for mix^2, this is due to the fact that $\gamma_{\bot\otimes\bot}(*,*) = -3$.

Let $\Delta = (\alpha_1 : \iota, \dots, \alpha_k : \iota)$ be a ground name context. Then an almost closed command c such that $\vdash c \mid \Delta$ has interpretation $[\![c]\!]_{(),\Delta} \in \mathbf{RelW}([\![\iota]\!]^{\mathsf{w}} \otimes \cdots \otimes [\![\iota]\!]^{\mathsf{w}}, \bot)$, that is $[\![c]\!]_{(),\Delta} \in \mathbf{RelW}(1, ?\mathsf{N} \parr \cdots \parr ?\mathsf{N})$. Let $m_1, \dots, m_k \in |?\mathsf{N}|$, then we have $\gamma_{?\mathsf{N}\parr\cdots\parr?\mathsf{N}}(m_1,\dots,m_k) = \gamma_{?N}(m_1)+\cdots+\gamma_{?N}(m_k)+k-1 = 2(\#m_1)-1+\cdots+2(\#m_k)-1+k-1 = 2(\#m_1+\cdots+\#m_k)-1$. So any element $(m_1,\dots,m_k)$ of $[\![c]\!]_{(),\Delta}$ must satisfy $2(\#m_1 + \cdots + \#m_k) - 1 = 1$, that is $\#m_1 + \cdots + \#m_k = 1$. Hence there must exist $i \in \{1,\dots,k\}$ such that $\#m_i = 1$ and $m_j = []$ for $j \neq i$. We retrieve semantically the fact that c is single valued.

7 Conclusion

We have developed a semantic investigation of classical PCF, presented in Herbelin's very pleasant $\overline{\lambda}\mu$ format. We have recalled the general LL semantic framework for this calculus, based on Girard's categorical semantics of LC, and its connection with Lafont-Reus-Streicher continuation categories. We have outlined a simple adequacy proof for the relational model and proposed a model which enforces uniqueness of values, rejecting the extension of classical PCF by a parallel composition construct based on the MIX rule of LL. In a longer version of this paper, we shall show that the Eilenberg-Moore category of the Scott semantics of LL admits a very simple description. The relational model of LL is also deeply related with useful extensions of LL (systems with bounded complexity, differential LL etc) which could suggest interesting extensions of classical PCF. For these reasons, we think that the LL-based semantics of classical PCF is worth being further studied.

Acknowledgments. We would like to thank the referees for their careful reading of this paper and their insightful remarks, as well as Thomas Streicher for a long and very useful discussion at an early stage of this work.

References

1 Pierre-Louis Curien and Hugo Herbelin. The duality of computation. In Martin Odersky and Philip Wadler, editors, *Proceedings of the Fifth ACM SIGPLAN International Confer-*

ence on Functional Programming (ICFP'00), Montreal, Canada, September 18-21, 2000., pages 233–243. ACM, 2000.

2 Vincent Danos and Jean-Louis Krivine. Disjunctive Tautologies as Synchronisation Schemes. In Peter Clote and Helmut Schwichtenberg, editors, *CSL*, volume 1862 of *Lecture Notes in Computer Science*, pages 292–301. Springer-Verlag, 2000.

3 Daniel De Carvalho. Execution Time of λ-Terms via Denotational Semantics and Intersection Types. Research Report RR-6638, INRIA, 2008. To appear in Mathematical Structures in Computer Sciences.

4 Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

5 Jean-Yves Girard. A new constructive logic: classical logic. *Mathematical Structures in Computer Science*, 1(3):225–296, 1991.

6 Jean-Yves Girard. *The blind spot: lectures on logic.* European Mathematical Society, 2011. 537 pages.

7 Timothy G. Griffin. The Formulae-as-Types Notion of Control. In *Proceedings of the 17th ACM Symposium on Principles of Programming Languages (POPL)*, pages 47–57. Association for Computing Machinery, January 1990.

8 Hugo Herbelin. Games and Weak-Head Reduction for Classical PCF. In Philippe de Groote, editor, *TLCA*, volume 1210 of *Lecture Notes in Computer Science*, pages 214–230. Springer-Verlag, 1997.

9 Martin Hofmann and Thomas Streicher. Completeness of Continuation Models for lambda-mu-Calculus. *Information and Computation*, 179(2):332–355, 2002.

10 Olivier Laurent. Polarized games. *Annals of Pure and Applied Logic*, 130(1-3):79–123, 2004.

11 Olivier Laurent and Laurent Regnier. About Translations of Classical Logic into Polarized Linear Logic. In *18th IEEE Symposium on Logic in Computer Science (LICS 2003), 22-25 June 2003, Ottawa, Canada, Proceedings*, pages 11–20. IEEE Computer Society, 2003.

12 Paul-André Melliès. Categorical Semantics of Linear Logic. *Panoramas et Synthèses*, 27, 2009.

13 Michel Parigot. $\lambda\mu$-Calculus: An Algorithmic Interpretation of Classical Natural Deduction. In Andrei Voronkov, editor, *Logic Programming and Automated Reasoning,International Conference LPAR'92, St. Petersburg, Russia, July 15-20, 1992, Proceedings*, volume 624 of *Lecture Notes in Computer Science*, pages 190–201. Springer-Verlag, 1992.

14 Peter Selinger. Control Categories and Duality: on the Categorical Semantics of the Lambda-Mu Calculus. *Mathematical Structures in Computer Science*, 11(2):207–260, 2001.

15 Thomas Streicher and Bernhard Reus. Classical Logic, Continuation Semantics and Abstract Machines. *Journal of Functional Programming*, 8(6):543–572, 1998.

16 Lionel Vaux. Convolution lambda-bar-mu-calculus. In Simona Ronchi Della Rocca, editor, *Typed Lambda Calculi and Applications, 8th International Conference, TLCA 2007, Paris, France, June 26-28, 2007, Proceedings*, volume 4583 of *Lecture Notes in Computer Science*, pages 381–395. Springer, 2007.

Uniform One-Dimensional Fragments with One Equivalence Relation

Emanuel Kieroński[1] and Antti Kuusisto[2]

1 **University of Wrocław, Poland,** kiero@cs.uni.wroc.pl
2 **Stockholm University, Sweden,** antti.j.kuusisto@gmail.com

Abstract

The uniform one-dimensional fragment U_1 of first-order logic was introduced recently as a natural generalization of the two-variable fragment FO^2 to contexts with relation symbols of all arities. It was shown that U_1 has the exponential model property and a NExpTime-complete satisfiability problem. In this paper we investigate two restrictions of U_1 that still contain FO^2. We call these logics RU_1 and SU_1, or the restricted and strongly restricted uniform one-dimensional fragments. We introduce Ehrenfeucht-Fraïssé games for the logics and prove that while SU_1 and RU_1 are expressively equivalent, they are strictly contained in U_1. Furthermore, we consider extensions of the logics SU_1, RU_1 and U_1 with unrestricted use of a single built-in equivalence relation $\sim$. We prove that while all the obtained systems retain the finite model property, their complexities differ. Namely, the satisfiability problem is NExpTime-complete for $SU_1(\sim)$ and 2-NExpTime-complete for both $RU_1(\sim)$ and $U_1(\sim)$. Finally, we show undecidability of some natural extensions of $SU_1(\sim)$.

1998 ACM Subject Classification F.4 Mathematical Logic and Formal Languages

Keywords and phrases two-variable logic, uniform one-dimensional fragments, complexity, expressivity, equivalence relations

Digital Object Identifier 10.4230/LIPIcs.CSL.2015.597

1 Introduction

Two-variable logic FO^2 was proved decidable in [17], and the satisfiability and finite satisfiability problems of FO^2 were shown NExpTime-complete in [7]. The extension of two-variable logic with counting quantifiers, FOC^2, was proved decidable in [8], [18]. It was subsequently shown to be NExpTime-complete in [20]. Research on extensions and variants of two-variable logic is *currently very active*. Recent research efforts have mainly concerned decidability and complexity issues over restricted classes of structures, and also questions related to different built-in features and operators that increase the expressivity of the base language. Recent articles in the field include for example [3, 11, 21, 23], and several others.

Typical systems of modal logic are contained in two-variable logic, or some variant of it, and hence investigations on two-variable logics have direct implications on various fields of computer science, including verification of software and hardware, distributed systems, knowledge representation and artificial intelligence. However, two-variable logics do not cope well with relations of arities greater than two, and therefore the *scope of related research is significantly restricted*. In database theory contexts, for example, two-variable logics as such are often not directly applicable due to the *severe arity-related limitations*.

Uniform one-dimensional fragment U_1 of first-order logic is a recently introduced formalism that generalizes two-variable logic to contexts with relation symbols of all arities. The fragment was originally defined in [9] and studied further in [10]. The fragment is based on restricting

24th EACSL Annual Conference on Computer Science Logic (CSL 2015).
Editor: Stephan Kreutzer; pp. 597–615

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

first-order logic in two ways. Firstly, quantification is restricted to blocks of existential (universal) quantifiers that *leave at most one free variable* in the resulting formula. Secondly, a *uniformity condition* applies to the use of atomic formulas: a Boolean combination of atoms $R(x_1, ..., x_k)$ and $S(y_1, ..., y_n)$, where $k, n \geq 2$, is allowed only if $\{x_1, ..., x_k\} = \{y_1, ..., y_n\}$. Boolean combinations of formulas with at most one free variable can be formed freely, and the use of equality is unrestricted.

It was established in [9] that if either of the two restrictions, one-dimensionality or uniformity, is lifted in a canonical way, the resulting formalism is undecidable. It was also established that already the equality-free fragment of U_1 can define properties not expressible in FOC^2 and also properties not expressible in the recently introduced *guarded negation fragment* [2], which significantly generalizes the *guarded fragment* [1] and *unary negation fragment* [14]. It was later established in [10] that U_1 has the finite model property and that the satisfiability problem of U_1 is NExpTime-complete. Thus the increase in expressivity when going from FO^2 to U_1 comes without cost in complexity. However, it was also proved in [10] that, in contrast to FO^2, adding counting quantifiers to U_1 leads to undecidability.

In this paper we investigate two restrictions of U_1 that still contain FO^2. We call these logics RU_1 and SU_1, or the *restricted* and *strongly restricted* uniform one-dimensional fragments. We begin our study by investigating the expressive power of the logics U_1, RU_1 and SU_1. We first provide Ehrenfeucht-Fraïssé game characterizations for our fragments; the rather simple and natural characterizations provide a nice algebraic perspective on the logics. We then establish that while SU_1 and RU_1 are expressively equivalent, they are strictly contained in U_1. Strictness of the containment follows by use of the EF-game for SU_1.

We then consider extensions of the logics SU_1, RU_1 and U_1 with a single built-in equivalence relation $\sim$ which can be used freely, i.e., the uniformity conditions do not apply to the use of $\sim$. We prove that while all the obtained systems retain the finite model property, their complexities differ. Namely, the satisfiability problem is NExpTime-complete for $SU_1(\sim)$ and 2-NExpTime-complete for both $RU_1(\sim)$ and $U_1(\sim)$. Thus we provide a complete classification of the complexities of the logics $SU_1(\sim)$, $RU_1(\sim)$ and $U_1(\sim)$.

We finish the investigations in this paper by establishing undecidability of some natural extensions of $SU_1(\sim)$. We show undecidability of the extension of SU_1 with *two* equivalence relations as well as the extension with one transitive relation. This contrasts with the case of FO^2 which remains decidable when extended by two equivalence relations [12, 13] or one transitive relation [23]. FO^2 with three equivalence relations is undecidable [12].

Built-in equivalence relations have played a visible role in recent investigations on two-variable logics, see for example [4, 5, 12, 13]. The articles [4, 5] discuss applications of two-variable logics with built-in equivalences in the context of *data words* and XML reasoning. In addition to being relevant in the context of data words, two-variable logics with equivalence relations naturally embed various different kinds of epistemic logics, where equivalence relations naturally correspond to epistemic indistinguishability relations of agents. Furthermore, the idea of adding equivalence relations in order to increase expressivity has been recently investigated in the context of interval temporal logics; see, e.g., [16].

Two-variable logics and guarded fragments are currently the two principal frameworks used for identifying decidable fragments of first-order logic. Originally, the logic U_1 was defined to be a generalization of FO^2, and in this respect U_1 is to FO^2 what the guarded negation fragment is to the guarded fragment—a reasonable generalization. U_1 has the same complexity as FO^2, and—as discussed above—its extension with counting quantifiers as well as its variants without either the uniformity or the one-dimensionality constraint, are undecidable. However, there are of course other decidable generalizations of FO^2, such

as FOC^2 and the novel logics RU_1 and SU_1. Hence it is important, we believe, to try to better understand the realm of decidable logics above FO^2. The investigations in this article contribute towards that aim. In particular, we observe, e.g., that the generalization SU_1 of FO^2 is of *lower complexity* than U_1 and RU_1 in the presence of a built-in equivalence.

2 Preliminaries

We let $\mathbb{Z}_+$ denote the set of positive integers and $\mathbb{N}$ the natural numbers. If $\overline{a}$ is a finite tuple of elements, we write $b \in \overline{a}$ in order to indicate that b is one of the elements of the tuple. By $(u, ..., u)$ we denote a finite tuple where each position contains the same element u; the arity of the tuple is unimportant or known from the context when this notation is used. We recall that $\bigwedge \emptyset = \top$ and $\bigvee \emptyset = \bot$. The order of priority of logical connectives when brackets are left unwritten is such that first come $\wedge$, $\vee$, and after that come $\rightarrow$, $\leftrightarrow$. The length of a formula φ is denoted by $\|\varphi\|$.

Let $\mathcal{V}$ denote a *complete relational vocabulary*, i.e., $\mathcal{V} := \bigcup_{k \in \mathbb{Z}_+} \tau_k$, where τ_k denotes a countably infinite set of k-ary relation symbols. Every vocabulary we consider below is assumed to be a subset of $\mathcal{V}$. In the sections concerning expressivity, we use the symbol σ in order to refer to *finite* vocabularies. In investigations concerning complexities of satisfiability problems, the vocabulary of the set of input formulas is always $\mathcal{V}$ extended with the special built-in symbols such as the equivalence relation symbol $\sim$. In this article a σ-model $\mathfrak{A}$ is a model that interprets at least the relation symbols in the vocabulary σ.

We let $\mathrm{VAR} = \{ v_i \mid i \in \mathbb{N} \}$ be the set of first-order variables. We mostly use *meta-variables* x, y, z, x_1, x_2, x_3, etc., in order to denote variables in VAR. We let $\mathit{diff}(x_1, ..., x_m)$ denote the conjunction $\bigwedge_{1 \leq i < j \leq m} x_i \neq x_j$. We define $\mathit{diff}(x) := x = x$.

Let $X = \{x_1, ..., x_m\} \neq \emptyset$ be a finite set of variable symbols. Let R be a k-ary relation symbol. if $\{x_{i_1}, ..., x_{i_k}\} = X$. Equalities $x = y$ are *not* $\{x, y\}$-atoms, since the definition requires a relation symbol to be used. A formula is called an *X-literal* if it is an X-atom or a negated X-atom.

Let τ be a vocabulary. A *k-ary τ-atom* is an atomic τ-formula ψ such that

$$|\{ x \in \mathrm{VAR} \mid \psi \text{ contains an instance of } x \}| \;=\; k.$$

For example, if $P \in \tau$ is a unary and $R \in \tau$ a ternary symbol, then $P(x)$, $x = x$, $R(x, x, x)$ are unary τ-atoms, and $R(v_1, v_2, v_2)$, $v_1 = v_2$ are binary τ-atoms.

The set of τ-formulas of the *uniform one-dimensional fragment* U_1 is the smallest set $\mathcal{F}$ satisfying the following conditions.

1. Every unary τ-atom is in $\mathcal{F}$. Also $\bot, \top \in \mathcal{F}$.
2. Every identity atom $x = y$ is in $\mathcal{F}$.
3. If $\varphi \in \mathcal{F}$, then $\neg\varphi \in \mathcal{F}$.
4. If $\varphi, \psi \in \mathcal{F}$, then $(\varphi \wedge \psi) \in \mathcal{F}$.
5. Let $Y := \{x_0, ..., x_k\} \subseteq \mathrm{VAR}$ and $X \subseteq Y$. Let φ be a Boolean combination of X-atoms over τ and formulas in $\mathcal{F}$ whose free variables (if any) are in Y. Then
   a. $\exists x_1 ... \exists x_k \varphi \;\in\; \mathcal{F}$ and
   b. $\exists x_0 ... \exists x_k \varphi \;\in\; \mathcal{F}$.

For example $\exists y \exists z \big(\neg Sxyz \wedge Py \wedge (Syzx \vee Tzyxz) \big)$ is a U_1-formula, while $\exists y \exists z (Rxy \wedge Ryz)$ is not. Now consider the U_1-formula $\exists y \exists z \big(x \neq y \wedge Ryz \big)$. The free variable x *does not* occur in the set $\{y, z\}$ that corresponds to the set X in clause 5 of the definition of U_1. Consider the clause 5.*a* which states that "$\exists x_1 ... \exists x_k \varphi \in \mathcal{F}$." Change the clause 5.*a* to the novel clause "if $x_0 \in X$, then $\exists x_1 ... \exists x_k \varphi \in \mathcal{F}$." The five clauses with this modified version of clause 5.*a*

define the set of τ-sentences RU_1. Note that in clause 5, the formula φ does not have to contain any X-atoms, so formulas such as $\exists y \exists z\big(x \neq y \wedge x \neq z\big)$ are in U_1 and RU_1.

Consider the RU_1-formula $\exists y \exists z\big(Rxy \wedge y \neq z\big)$. The free variable z is not in the set $\{x, y\}$ which corresponds to the set X in clause 5 of the defintion of U_1. Consider the variant of the clause 5.*a* stating that "if $x_0 \in X$ and $X = \{x_0, ..., x_k\}$, then $\exists x_1 ... \exists x_k \varphi \in \mathcal{F}$." Consider also a variant of the rule 5.*b* which states that "if $X = \{x_0, ..., x_k\}$, then $\exists x_0 ... \exists x_k \varphi \in \mathcal{F}$." The five clauses with these modified versions of 5.*a* and 5.*b* define the set of τ-sentences of SU_1.

The above minor modifications to the syntax of U_1 that lead to RU_1 and SU_1 deal with *the way free and bound variables of formulas interact with relation symbols of higher arities*. The modifications lead to interesting complexity issues, as we will see. Our Ehrenfeucht-Fraïssé characterizations show that the (initially perhaps somewhat complicated) logics correspond to natural algebraic back and forth conditions that extend the well-known two-pebble games for FO^2. It is worth noting that clearly even the weakest of our logics, SU_1, contains FO^2.

We then define extensions of the three logics U_1, RU_1, SU_1 by a single built-in equivalence relation $\sim$. A formula φ is a τ-formula of $\mathrm{U}_1(\sim)$ if and only if it can be obtained from some τ-formula of U_1 by replacing any number of equality symbols $=$ by the equivalence symbol $\sim$. The logics $\mathrm{RU}_1(\sim)$ and $\mathrm{SU}_1(\sim)$ are defined analogously from RU_1 and SU_1.

We define the *quantifier block rank* of a U_1-formula φ, or $qbr(\varphi)$, as follows.

1. $qbr(\varphi) = 0$ iff φ is quantifier-free.
2. $qbr(\varphi \wedge \psi) = max(qbr(\varphi), qbr(\psi))$; $qbr(\neg\varphi) = qbr(\varphi)$.
3. Assume $\varphi := \exists \overline{x}\, \chi$, where χ does *not* begin with $\exists$. Then $qbr(\varphi) = qbr(\chi) + 1$.

We define the *quantifier width* of a U_1-formula φ, or $qw(\varphi)$, as follows.

1. $qw(\varphi) = 1$ iff φ is atomic and has a free variable. $\top$ and $\bot$ have quantifier width 0.
2. $qw(\varphi \wedge \psi) = max(qw(\varphi), qw(\psi))$; $qw(\neg\varphi) = qw(\varphi)$.
3. Assume $\varphi := \exists x_1 ... \exists x_k\, \chi$, where χ does *not* begin with $\exists$. If φ has a free variable, then $qw(\varphi) = max(1 + k, qw(\chi))$. If φ is a sentence, then $qw(\varphi) = max(k, qw(\chi))$.

The pair $(qbr(\varphi), qw(\varphi))$ is the *rank* of a U_1-formula. Note that SU_1 and RU_1 are fragments of U_1, so various technical definitions, such as the above definition of a notion of rank, automatically concern SU_1 and RU_1 as well.

Let $\overline{x}$ denote a tuple of variables. Let $\chi := \exists \overline{x}\, \varphi$ be a U_1-formula formed by using the formula construction rule 5. Assume φ is quantifier-free. Then we call φ a U_1-*matrix*. If φ does not contain k-ary atoms for any $k \geq 2$, with the possible exception of equality atoms $x = y$, then we define $S_\varphi := \emptyset$. Otherwise we define S_φ to be the set X used in the construction of χ (see rule 5). The set S_φ is the set of *live variables* of φ. Let $\psi(x_0, ..., x_k)$ be a U_1-matrix, where $(x_0, ..., x_k)$ enumerates the variables of ψ. Let $\mathfrak{A}$ be a structure and $a_0, \ldots, a_k \in A$. We let $\mathrm{live}\big(\psi(x_0, ..., x_k)[a_0, \ldots, a_k]\big)$ denote the smallest set $T \subseteq \{a_0, ..., a_k\}$ such that $a_i \in T$ if x_i is a live variable of $\psi(x_0, ..., x_k)$. We may write $\mathrm{live}\big(\psi[a_0, ..., a_k]\big)$ instead of $\mathrm{live}\big(\psi(x_0, ..., x_k)[a_0, \ldots, a_k]\big)$ when no confusion can arise. Notice that since the elements a_i are not required to be distinct, it is possible that $|\mathrm{live}(\psi[a_0, ..., a_k])|$ is smaller than the number of live variables in ψ.

2.1 Normal form and types

We introduce a normal form for our uniform one-dimensional logics which is inspired by the Scott normal form for FO^2 [22]. We say that a $\mathrm{U}_1(\sim)$ ($\mathrm{SU}_1(\sim)$, $\mathrm{RU}_1(\sim)$) formula φ is in *generalized Scott normal form* if φ has the following shape

$$\bigwedge_{1 \leq i \leq m_\exists} \forall x \exists y_1 \ldots y_{k_i} \varphi_i^\exists \wedge \bigwedge_{1 \leq i \leq m_\forall} \forall x_1 \ldots x_{l_i} \varphi_i^\forall, \tag{1}$$

where $\varphi_i^\exists = \varphi_i^\exists(x, y_1, \ldots, y_{k_i})$ and $\varphi^\forall = \varphi_i^\forall(x_1, \ldots, x_{l_i})$ are quantifier-free. The following proposition is a natural generalisation of Proposition 1 in [10]. It can be proved in the standard fashion, see, e.g., [6].

▶ **Proposition 1.** For every $\mathrm{U}_1(\sim)$ *(*$\mathrm{SU}_1(\sim)$, $\mathrm{RU}_1(\sim)$*)* formula φ, one can compute in polynomial time a $\mathrm{U}_1(\sim)$ *(*$\mathrm{SU}_1(\sim)$, $\mathrm{RU}_1(\sim)$*)* formula φ' in generalized Scott normal form (over a signature extended by some fresh unary symbols) such that φ and φ' are satisfiable over the same domains. Any model of φ can be expanded to a model of φ' by defining new unary symbols. Any model of φ' restricted to the signature of φ is a model of φ.

Let φ be a $\mathrm{U}_1(\sim)$ formula in generalized normal form, let $\mathfrak{A} \models \varphi$, $a \in A$, and let $b_1, \ldots, b_{k_i}$ be such that $\mathfrak{A} \models \varphi_i^\exists[a, b_1, \ldots, b_{k_i}]$. We say that $\mathfrak{B} = \mathfrak{A}\restriction\{a, b_1, \ldots, b_{k_i}\}$ (i.e., the restriction of $\mathfrak{A}$ to $\{a, b_1, \ldots, b_{k_i}\}$) is a *witness structure* for a and $\varphi_i^\exists$. The substructure of $\mathfrak{B}$ restricted to the elements of $\mathrm{live}(\varphi_i^\exists[a, b_1, \ldots, b_{k_i}])$ is called the *live part* of $\mathfrak{B}$. If the live part of $\mathfrak{B}$ does not contain a, then it is called *free*. Note that $|B|$ may be smaller than $k_i + 1$. Also, a may be a member of the live part of $\mathfrak{B}$ even if the variable x is not live in $\varphi_i^\exists$.

Let σ be a finite vocabulary. Let $\mathfrak{B}$ be a σ-model. Let $k \geq 1$ be an integer and $\bar{b} = (b_1, ..., b_k) \in B^k$ a tuple of *distinct* elements of $\mathfrak{B}$. Let $X = \{x_1, ..., x_k\}$ be a set of k *distinct* variables. Let T be the set of exactly all X-literals $\varphi(x_1, ..., x_k)$ over σ such that $\mathfrak{B} \models \varphi(b_1, ..., b_k)$. The conjunction $\bigwedge T$ is the *diagram type* of $\mathfrak{B}, \bar{b}$ over σ and with respect to the tuple $(x_1, ..., x_k)$. We denote this formula by $\delta_\sigma^{\mathfrak{B},\bar{b}}(x_1, ..., x_k)$. We assume some standard syntactic form (ordering of conjuncts and bracketing), so that if two formulas $\delta_\sigma^{\mathfrak{A},\bar{a}}(x_1, ..., x_k)$ and $\delta_\sigma^{\mathfrak{B},\bar{b}}(x_1, ..., x_k)$ are equivalent, they are one and the same formula.

Let $\tau \subseteq \mathcal{V}$ be a finite vocabulary. A 1-*type* over τ is a maximal satisfiable set of literals (atoms and negated atoms) over τ in the variable v_1. The set of all 1-types over τ is denoted by $\boldsymbol{\alpha}[\tau]$, or just by $\boldsymbol{\alpha}$ when τ is clear.

We identify 1-types α and conjunctions $\bigwedge \alpha$. A k-*table* over τ is a maximal satisfiable set of $\{v_1, ..., v_k\}$-atoms and negated $\{v_1, ..., v_k\}$-atoms over τ. Recall that a $\{v_1, ..., v_k\}$-atom must contain exactly the variables in $\{v_1, ..., v_k\}$, and note that a 2-table contains neither equality formulas nor negated equality formulas. We identify k-tables β and conjunctions $\bigwedge \beta$. We note that k-tables and diagram types are closely related notions.

Let $\mathfrak{A}$ be a τ-structure, and let $a \in A$. Let α be a 1-type over τ. We say that a *realizes* α if α is the unique 1-type such that $\mathfrak{A} \models \alpha[a]$. We let $\mathrm{tp}_\mathfrak{A}(a)$ denote the 1-type realized by a. Similarly, for *distinct* elements $a_1, \ldots, a_k \in A$, we let $\mathrm{tb}_\mathfrak{A}(a_1, \ldots, a_k)$ denote the unique k-table *realized* by the tuple $(a_1, \ldots, a_k)$, i.e., the k-table $\beta(v_1, ..., v_k)$ such that $\mathfrak{A} \models \beta[a_1, \ldots, a_k]$. Note that we have $\mathrm{tp}_\mathfrak{A}(a) \equiv \mathrm{tb}_\mathfrak{A}(a)$ for every $a \in A$.

Let us further introduce some new helpful terminology. A *multitype* is a function $\boldsymbol{\alpha} \to \mathbb{N}$. We say that a multitype θ is a k-*multitype* if $\sum_{\alpha \in \boldsymbol{\alpha}} \theta(\alpha) = k$. For a given set $\{a_1, \ldots, a_k\}$ of distinct elements from a structure $\mathfrak{A}$, we say that they *realize* a k-multitype θ, if for each $\alpha \in \boldsymbol{\alpha}$, we have that $\theta(\alpha)$ is the number of elements in $\{a_1, \ldots, a_k\}$ of 1-type α. If $\mathfrak{A}$ interpretes an equivalence relation $\sim$, then we say that a multitype is realized *in a class* D if it is realized by a subset of elements of the equivalence class D of $\mathfrak{A}$. We say that a multitype is realized *by a class* if it is realized by the set of all elements of this equivalence class.

3 Games for U_1 and SU_1

In this section we provide Ehrenfeucht-Fraïssé game characterizations for U_1 and SU_1. A similar characterization exists for RU_1, but we will not discuss it explicitly.

We will below define the games rigorously, but *roughly*, the game for U_1 involves positions encoded by a bijection between finite subsets of two models. *Spoiler* chooses (at most) one

pair (u, u') of bijectively related points. Then he chooses a finite *blue* set B and a *green* set $G \subseteq B$ from one of the models such that we have $u \in B$ or $u' \in B$, depending on which model B was chosen from. *Duplicator* responds by a *new* bijection from B onto a subset of the other model. Intuitively, the bijection defines counterparts of the blue and green sets in the other model. Information about the relations of the two models in restriction to the sets B, G and their counterparts in the other model, are then compared (in a way specified later).

Let σ be a finite vocabulary. Let $\mathfrak{A}$ and $\mathfrak{D}$ be σ-models. Let $k \in \mathbb{N}$ and $n \in \mathbb{Z}_+$. Let $S \subseteq A$ and $T \subseteq D$ be *finite* (possibly empty) sets such that $|S| = |T|$. Let $f : S \to T$ be a bijection. We next define the game $G^{k,n}_\sigma(\mathfrak{A}, S, f, \mathfrak{D}, T)$ that characterizes expressivity of U_1-formulas of rank (k, n) and over σ.

The game is played between two players, Spoiler and Duplicator. The game begins from the *position* $(\mathfrak{A}, S_k, f_k, \mathfrak{D}, T_k)$, where $S_k = S$, $T_k = T$ and $f_k = f$. If $k = 0$, the (play of the) game ends immediately in the beginning position $(\mathfrak{A}, S, f, \mathfrak{D}, T)$. If $k \neq 0$, the game is played for k *rounds*; the game begins with round k, and each round $j \neq 0$ is followed by round $j - 1$. Round $j \in \{1, ..., k\}$ begins from a position denoted by $(\mathfrak{A}, S_j, f_j, \mathfrak{D}, T_j)$ and ends in a position $(\mathfrak{A}, S_{j-1}, f_{j-1}, \mathfrak{D}, T_{j-1})$, and the game ends in a position $(\mathfrak{A}, S_0, f_0, \mathfrak{D}, T_0)$; for each $j \in \{0, ..., k\}$, we have $S_j \subseteq A$ and $T_j \subseteq D$, while f_j is a bijection from S_j onto T_j. Round $j \in \{1, ..., k\}$ consists of a move by Spoiler and a response by Duplicator. These actions determine how the positions of the game arise and evolve.

In round j, Spoiler first decides whether he wants to make a *local* or a *global* move. Assume first that he decides upon a local move. Local moves are allowed only when S_j and T_j are nonempty. Spoiler chooses one of the pairs $\mathfrak{A}, S_j$ and $\mathfrak{D}, T_j$. Let us assume he chooses $\mathfrak{A}, S_j$. Spoiler then chooses an element $r \in S_j$ and sets $B \subseteq A$ and $G \subseteq B$ such that $|B| \leq n$ and $r \in B$. We call r the *red* element coloured by Spoiler in round j, and we call B and G the sets of *blue* and *green* elements coloured by Spoiler in round j. (Note that green elements are blue as well, and the red element r must be blue and can be green.) Once Spoiler has appointed the element r and the sets G and B, Duplicator chooses an injection $h : B \to D$ such that $h(r) = f_j(r)$. The game continues from the position $(\mathfrak{A}, S_{j-1}, f_{j-1}, \mathfrak{D}, T_{j-1}) := (\mathfrak{A}, B, h, \mathfrak{D}, h(B))$. (We define $h(B) = \{\, h(b) \mid b \in B \,\}$.)

If Spoiler chooses the pair $\mathfrak{D}, T_j$ instead of $\mathfrak{A}, S_j$, the rules of the game are symmetric; Spoiler chooses a red element $r' \in T_j$ and blue and green sets $B' \subseteq D$ and $G' \subseteq B$ such that $|B'| \leq n$ and $r' \in B'$. Duplicator responds by an injection $h : B' \to A$ such that $h(r') = f_j^{-1}(r')$, where f_j^{-1} denotes the inverse function of the bijection f_j. The inverse function of the injection h is the novel bijection f_{j-1} from the novel set $S_{j-1} := h(B')$ onto the blue set B'. Of course $T_{j-1} := B'$.

If Spoiler decides upon a *global* move instead of a local one, he first chooses one of the structures $\mathfrak{A}$ and $\mathfrak{D}$. Let us assume that he chooses $\mathfrak{A}$. Spoiler then chooses a blue set $B \subseteq A$ and a green set $G \subseteq B$ such that $|B| \leq n$. Duplicator responds by an injection $h : B \to D$. The game continues from the position $(\mathfrak{A}, S_{j-1}, f_{j-1}, \mathfrak{D}, T_{j-1}) := (\mathfrak{A}, B, h, \mathfrak{D}, h(B))$. Again if Spoiler chooses the structure $\mathfrak{D}$ instead of $\mathfrak{A}$, he chooses the blue and green sets B' and G' from $\mathfrak{D}$. Duplicator then responds by an injection h from B' into A. The inverse function of h becomes the bijection f_{j-1}. Of course $|B'| \leq n$ and $G' \subseteq B'$.

We then describe the winning conditions of the game. We begin with some auxiliary definitions. Let X be a set, and let $l \in \mathbb{Z}_+$. Let $(u_1, ..., u_l) \in X^l$ be a tuple and $Y \subseteq X$. We say that $(u_1, ..., u_l)$ *spans* the set Y if $\{u_1, ..., u_l\} = Y$. Note that it is possible that $(u_1, ..., u_l)$ spans Y even if $|Y| < l$.

Let $\mathfrak{A}$ and $\mathfrak{D}$ be σ-structures. Let $G \subseteq A$ and $G' \subseteq D$ be finite sets. Let f be a bijection from G onto G'. We say that f *preserves spanning tuples* over σ and write $\mathfrak{A}, G \ \langle f, \sigma \rangle \ \mathfrak{D}, G'$,

if for each symbol $R \in \sigma$ and each tuple $\overline{a}$ that spans G, we have $\overline{a} \in R^{\mathfrak{A}} \Leftrightarrow f(\overline{a}) \in R^{\mathfrak{D}}$. (We define $f(\overline{a}) = (f(a_1), ..., f(a_p))$, where $\overline{a} = (a_1, ..., a_p)$.)

Duplicator wins a play of the game $G^{k,n}_{\sigma}(\mathfrak{A}, S, f, \mathfrak{D}, T)$ iff the conditions below hold.

1. Consider round $j \in \{1, ..., k\}$ of the game. If Spoiler makes his moves in $\mathfrak{A}$, then let $G \subseteq A$ be the green set coloured by Spoiler in round j. If Spoiler makes his moves in $\mathfrak{D}$, let G be the set $h(G')$, where $G' \subseteq D$ is the green set coloured by Spoiler in round j and h is the injection chosen by Duplicator. The restriction of f_{j-1} to G preserves spanning tuples over σ, i.e., $\mathfrak{A}, G \ \langle f_{j-1} \restriction G, \sigma \rangle \ \mathfrak{D}, f_{j-1}(G)$.
2. Recall that $(a, ..., a)$ denotes a tuple where each coordinate position contains a. Let $j \in \{0, ..., k\}$. For all $R \in \sigma$ and all $a \in S_j$, we have $(a, ..., a) \in R^{\mathfrak{A}} \Leftrightarrow (f_j(a), ..., f_j(a)) \in R^{\mathfrak{D}}$. In particular, $a \in P^{\mathfrak{A}} \Leftrightarrow f_j(a) \in P^{\mathfrak{D}}$ for each unary symbol $P \in \sigma$ and each $a \in S_j$.

We write $\mathfrak{A}, S \sim^{k,n}_{f,\sigma} \mathfrak{D}, T$ if *Duplicator* has a winning strategy in the game. A strategy of Duplicator is simply a function that takes as an argument a position in the game together with a move of Spoiler in that position; the value of the function with such an input is a specification of the response move of Duplicator. A strategy is a winning strategy if it guarantees a win in every play of the game.

Now consider a variant $\hat{G}^{k,n}_{\sigma}(\mathfrak{A}, S, f, \mathfrak{D}, T)$ of the game $G^{k,n}_{\sigma}(\mathfrak{A}, S, f, \mathfrak{D}, T)$ defined by adding to the game $G^{k,n}_{\sigma}(\mathfrak{A}, S, f, \mathfrak{D}, T)$ the additional rule that the green set chosen by Spoiler must always be either empty or equal to the blue set. In other words, if B and G are the blue and green sets chosen in some round of the game, then we have $G \in \{\emptyset, B\}$. Let $\hat{\sim}^{k,n}_{f,\sigma}$ denote the relation analogous to $\sim^{k,n}_{f,\sigma}$ but concerning the new variant of the game.

Let $\mathfrak{A}$ and $\mathfrak{D}$ be σ-models. Let $S \subseteq A$ and $T \subseteq D$ be equicardinal finite sets, and let $f : S \to T$ be a bijection. We write $\mathfrak{A}, S \equiv^{k,n}_{f,\sigma} \mathfrak{B}, T$ if the equivalence $\mathfrak{A} \models \varphi(\overline{a}) \Leftrightarrow \mathfrak{D} \models \varphi(f(\overline{a}))$ holds for all tuples $\overline{a}$ of elements of S and all U_1-formulas $\varphi(\overline{x})$ of rank (k, n) over σ. We let $\hat{\equiv}^{k,n}_{f,\sigma}$ denote the relation analogous to $\equiv^{k,n}_{f,\sigma}$ but concerning SU_1-formulas instead of U_1-formulas. When σ is clear or irrelevant, we may leave it unwritten.

The following theorem is relatively easy but tedious to prove. A detailed proof will be presented in the full version of the paper.

▶ **Theorem 2.** $\mathfrak{A}, S \equiv^{k,n}_{f} \mathfrak{D}, T \Leftrightarrow \mathfrak{A}, S \sim^{k,n}_{f} \mathfrak{D}, T$ *and* $\mathfrak{A}, S \hat{\equiv}^{k,n}_{f} \mathfrak{D}, T \Leftrightarrow \mathfrak{A}, S \hat{\sim}^{k,n}_{f} \mathfrak{D}, T$.

4 Comparing the expressive power

In this section we first establish that SU_1 is strictly less expressive than U_1.

Let R be a ternary relation. The U_1-sentence

$$\exists v \forall x \forall y \forall z \big(Rxyz \to (v = x \vee v = y \vee v = z) \big)$$

states that some v belongs to every tuple of R. Let us call this the *covering node property*.

▶ **Theorem 3.** *The covering node property is not expressible in* SU_1.

Proof. We begin by defining two models $\mathfrak{M}$ and $\mathfrak{N}$ with a *ternary* relation R. Intuitively, both of these models represent a hypergraph where each edge has exactly three elements. We define the model $\mathfrak{M} = (M, R^{\mathfrak{M}})$ such that $M = \{0, 1, 2, 3, 4, 5, 6\}$ and for each $(u, v, w) \in M^3$, we have $(u, v, w) \in R^{\mathfrak{M}}$ iff $\{u, v, w\} \in \{ \{0, 1, 2\}, \{0, 3, 4\}, \{0, 5, 6\} \}$. We define the model $\mathfrak{N} = (N, R^{\mathfrak{N}})$ such that $N = \{a, b, c, d, e, f, g\}$, and for each $(u, v, w) \in N^3$, we have $(u, v, w) \in R^{\mathfrak{N}}$ iff $\{u, v, w\} \in \{ \{a, b, c\}, \{c, d, e\}, \{e, f, g\} \}$. We note that while $\mathfrak{M}$ satisfies the covering node property, $\mathfrak{N}$ does not.

We then fix some terminology for later use. Let $\mathfrak{A} = (A, R^{\mathfrak{A}})$ be a model that represents a hypergraph where each edge has exactly three elements, meaning that for each $(u, v, w) \in A^3$, if $(u, v, w) \in R^{\mathfrak{A}}$, then every permutation of the tuple (u, v, w) is also in $R^{\mathfrak{A}}$ and $|\{u, v, w\}| = 3$. Let $t \in A$. We say that t is *incident to an edge* if we have $(t, t', t'') \in R^{\mathfrak{A}}$ for some elements $t', t'' \in A$. We say that t is *incident to a gap* if there exist elements $t', t'' \in A$ such that $(t, t', t'') \notin R^{\mathfrak{A}}$ and $|\{t, t', t''\}| = 3$. A subset S of A is an *edge* iff $S = \{s, s', s''\}$ for some elements s, s', s'' such that $(s, s', s'') \in R^{\mathfrak{A}}$.

Fix an arbitrary pair (k, n); we will show that $\mathfrak{M}, \emptyset \mathrel{\hat{\equiv}}^{k,n}_{\emptyset} \mathfrak{N}, \emptyset$ by using the game for SU_1.

Assume first a position $(\mathfrak{M}, S, f, \mathfrak{N}, T)$ has been reached in the game. We show how Duplicator plays in that position.

Assume Spoiler chooses a blue set B and a green set $G \in \{B, \emptyset\}$. If Spoiler is making a local move, he also chooses a red element r which is in $S \cap B$ if Spoiler moves in $\mathfrak{A}$ and in $T \cap B$ otherwise. If $|B| \neq 3$, Duplicator responds by choosing an *arbitrary* injection h that maps the elements of B into the other model, and if Spoiler has made a local move, then also $h(r) = f(r)$ or $h(r) = f^{-1}(r)$ holds. Duplicator can always do this since $|M| = |N|$.

If $|B| = 3$, then the move of Duplicator depends on whether B is an edge. We assume that $G = B$. (In the case where $G = \emptyset$, Duplicator acts precisely the same way as in the case $G = B$.) Duplicator must choose an injection h' such that $h'(B)$ is an edge iff also B is an edge. Furthermore, if Spoiler has made a local move and thus appointed a red element r', which is necessarily in $S \cap B$ or in $T \cap B$ (depending on which model Spoiler moves in), the injection h' must map r' to $f(r')$ or $f^{-1}(r')$. Duplicator can always choose such an injection h' since *in both models, each element is incident to an edge as well as a gap.* ◀

On the other hand, it turns out that RU_1 and SU_1 have the same expressive power.

▶ **Theorem 4.** RU_1 *and* SU_1 *are expressively equivalent.*

Proof. Let σ be the vocabulary of a formula to be translated. It is easy to show that σ-formulas of RU_1 can be represented in a normal form where each formula $\exists \overline{x}\, \varphi$ is such that φ has the following shape

$$\delta(x_1, ..., x_q) \wedge \mathit{diff}(x_1, ..., x_r) \wedge \bigwedge_{i \in \{1,...,r\}} \tau_i(x_i),$$

where $\delta(x_1, ..., x_q)$ is a diagram type, $q \leq r$, and the formulas $\tau_i(x_i)$ are so-called *types of rank* (k, n). Types of rank (k, n) have the property that for each i and $j \neq i$, either $\tau_i(y)$ and $\tau_j(y)$ are equivalent, or the conjunction $\tau_i(y) \wedge \tau_j(y)$ is not satisfiable. Types of rank (k, n) have various analogous incarnations in various different contexts of finite model theory; see for example [15] for *rank-k types* for FO and also similar types for finite variable logics.

We define a translation t from such normal form formulas into SU_1 such that $t(\varphi) = \varphi$ for atoms and $t(\varphi \wedge \psi) = t(\varphi) \wedge t(\psi)$ and $t(\neg\varphi) = \neg t(\varphi)$. Formulas $\exists \overline{x}\, \varphi$ are trickier to translate. Let $\chi(x_1) := \exists x_2 ... \exists x_r \psi$, where ψ is the formula $\delta(x_1, ..., x_q) \wedge \mathit{diff}(x_1, ..., x_r) \wedge \bigwedge_{i \in \{1,...,r\}} \tau_i(x_i)$. Let us translate $\chi(x_1)$ into SU_1. Define $\chi'(x_1)$ to be the formula

$$\exists x_2 ... \exists x_q \big(\delta(x_1, ..., x_q) \wedge \mathit{diff}(x_1, ..., x_q) \wedge \bigwedge_{i \in \{1,...,q\}} \tau_i(x_i)\big) \wedge \exists x_2 ... \exists x_r \big(\mathit{diff}(x_1, ..., x_r) \wedge \bigwedge_{i \in \{1,...,r\}} \tau_i(x_i)\big).$$

(Notice carefully how the indices q and r are now placed.) Recalling the properties of the formulas $\tau_i(x_i)$ discussed above, it is easy to see that $\chi'(x_1)$ is equivalent to $\chi(x_1)$. The

translation $t(\chi(x_1))$ is obtained from $\chi'(x_1)$ by replacing the conjuncts $\tau_i(x_i)$ in the above conjunctions by $t(\tau_i(x_i))$.

Now consider a formula $\gamma := \exists \overline{x}\, \psi$ *without* free variables. Remove one variable y from $\overline{x}$ and let $\eta(y)$ denote the obtained formula; the variable y is assumed to be part of the diagram type in ψ. Now obtain from $\eta(y)$ the formula $t(\eta(y))$ in the way $t(\chi(x_1))$ was obtained from $\chi(x_1)$ above. We define $t(\gamma) := \exists y\, t(\eta(y))$. ◀

5 Built-in equivalence relations and complexity

It is not difficult to show (e.g., using the games we introduced in this paper) that uniform one-dimensional logics cannot express that a binary relation is an equivalence. In this section we consider the logics $\mathrm{U}_1(\sim)$, $\mathrm{RU}_1(\sim)$, $\mathrm{SU}_1(\sim)$ that have free use of the equivalence relation $\sim$. (An alternative, but less interesting and less expressive variant would allow only uniform use of $\sim$, i.e., the use of $\sim$ as if it was an ordinary binary relation symbol.)

Even in the simplest of the three logics, $\mathrm{SU}_1(\sim)$, one can express pretty complex properties such as, e.g., the existence of at most two equivalence classes with more than two elements:

$$\forall x_1 ... x_9 \big(\mathit{diff}(x_1, \ldots, x_9) \wedge (x_1 \sim x_2 \sim x_3) \wedge (x_4 \sim x_5 \sim x_6) \wedge x_1 \not\sim x_4 \\ \rightarrow x_7 \sim x_1 \vee x_7 \sim x_4 \vee x_7 \not\sim x_8 \vee x_7 \not\sim x_9 \big).$$

Unrestricted use of $\sim$ allows also for a non-trivial interaction of $\sim$ with relations of arity greater than 2. One can express, e.g, that if, say, four elements are connected by a four-ary predicate R, then they are members of at least two equivalence classes.

We first observe that in $\mathrm{RU}_1(\sim)$, models of doubly exponential size can be enforced, and use this to show a 2-NExpTime-lower bound for the satisfiability problem. Then we show that all our logics have the exponential classes property: if a formula is satisfiable, then it has a model in which all equivalence classes are bounded exponentially. We further use this result to show that $\mathrm{SU}_1(\sim)$ has the exponential model property, and that both $\mathrm{RU}_1(\sim)$ and $\mathrm{U}_1(\sim)$ have the doubly exponential model property. This leads to tight complexity bounds for each logic.

5.1 Lower bound for $\mathrm{RU}_1(\sim)$

In this section we show that the satisfiability and finite satisfiability problems for $\mathrm{RU}_1(\sim)$ (and thus also for $\mathrm{U}_1(\sim)$) are 2-NExpTime-hard. In particular this demonstrates that in $\mathrm{RU}_1(\sim)$ one can construct satisfiable formulas whose models are of at least doubly exponential size with respect to their length.

We employ a reduction from a variant of the tiling problem. Let $\mathfrak{G}_m$ denote the standard $m \times m$ grid, $\mathfrak{G}_m = ([0, m-1]^2, H, V)$ with the horizontal and vertical successor relations H and V. A *tiling system* is a quadruple $\mathcal{T} = \langle C, c_0, \mathit{Hor}, \mathit{Ver} \rangle$, where C is a non-empty, finite set of *colours*, c_0 is an element of C, and Hor, Ver are binary relations on C called the *horizontal* and *vertical* constraints, respectively. A *tiling* for $\mathcal{T}$ of a grid $\mathfrak{G}_m$ is a function $f : G_m \to C$ such that $f(0,0) = c_0$, and for all $(d, d') \in H$, the pair $\langle f(d), f(d') \rangle$ is in Hor, and for all $(d, d') \in V$, the pair $\langle f(d), f(d') \rangle$ is in Ver. The *doubly exponential tiling problem* consists in checking for a given $n \in \mathbb{N}$ written in unary, and a tiling system $\mathcal{T}$, if $\mathcal{T}$ has a tiling of the grid $\mathfrak{G}_m$, where $m = 2^{2^n}$. It is well known that the doubly exponential tiling problem is 2-NExpTime-complete (see, e.g., [19], p. 501).

▶ **Theorem 5.** *The satisfiability and the finite satisfiability problems for* $\mathrm{RU}_1(\sim)$ *are hard for 2-*NExpTime*.*

The proof is similar in spirit to the proof of the 2-NExpTime-lower bound for the two-variable fragment with two equivalence relations given in [11]. The crux is a succinct axiomatization of a grid structure of doubly exponential size.

Let $U_0, \ldots, U_{n-1}$ be unary predicates. By taking the predicates U_i to indicate the values of binary digits, we may take each element in any structure interpreting these predicates to have a 'local coordinate' in the range $[0, 2^n - 1]$; a point u of a model encodes the binary string s such that the ith bit of s is 1 iff $U_i(u)$ holds. It helps to think that an element's local coordinate fixes its position inside its equivalence class. We employ the abbreviation $\lambda^{=}(x, y)$ in order to state that x and y (which may be from different classes) have the same local coordinates; $\lambda^{<}(x, y)$ to state that the local coordinate of y is greater than the local coordinate of x; and $\lambda^{+1}(x, y)$ to state that the local coordinate of y is one greater than the local coordinate of x (addition modulo 2^n). All these abbreviations can be defined in the standard way using quantifier-free formulas of length polynomial in n. The formula

$$\forall x \exists y \big(x \sim y \wedge \lambda^{+1}(x, y)\big) \tag{2}$$

then ensures that each class contains a collection of 2^n elements, distinguished by local coordinates in the range $[0, 2^n - 1]$.

We now endow each class with a pair of 'global coordinates', corresponding to the grid coordinates in the range $[0, 2^{2^n} - 1]$. Let X and Y be unary predicates. The conjunct

$$\forall x y \big(x \sim y \wedge \lambda^{=}(x, y) \to ((X(x) \leftrightarrow X(y)) \wedge (Y(x) \leftrightarrow Y(y)))\big) \tag{3}$$

ensures that elements of the same class with the same local coordinates agree on the satisfaction of X and Y. For simplicity we allow ourselves to speak of *the* element of some class with a given local coordinate, since all such elements will turn out to have identical properties. If D is a class, we take the global X-coordinate of D to be the number in the range $[0, 2^{2^n} - 1]$ whose jth bit ($0 \leq j \leq 2^n - 1$) is 1 iff the element of D whose local coordinate is j satisfies the predicate X. Likewise, we define the global Y-coordinate of D using the predicate Y.

Now we enforce that for a class with global coordinates (p, q), there exists a class with coordinates $(p + 1, q)$ (if $p < 2^{2^n} - 1$) and a class with coordinates $(p, q + 1)$ (if $q < 2^{2^n} - 1$).

We take the predicate X^1 to mark in each class the least significant position satisfying X, and we define X^0 symmetrically:

$$\forall x \big(X^1(x) \leftrightarrow (X(x) \wedge \forall y(x \sim y \wedge \lambda^{<}(y, x) \to \neg X(y)))\big), \tag{4}$$

$$\forall x \big(X^0(x) \leftrightarrow (\neg X(x) \wedge \forall y(x \sim y \wedge \lambda^{<}(y, x) \to X(y)))\big). \tag{5}$$

Consider now the following formulas.

$$\forall x \big(X^0(x) \to \exists y(X^1(y) \wedge \lambda^{=}(x, y) \wedge H(x, y))\big), \tag{6}$$

$$\forall x y x' y' \big(x \sim y \wedge x' \sim y' \wedge \lambda^{=}(x, x') \wedge H(y, y') \to ((Y(x) \leftrightarrow Y(x')) \wedge (\lambda^{<}(y, x) \to (X(x) \leftrightarrow X(x'))))\big) \tag{7}$$

They link via H the element marked by X^0 from one class to the element marked by X^1 from another class with the X-coordinate greater by one and with the same Y-coordinate.

Let (8)–(11) be formulas analogous to (4)-(7) using predicates Y^1, Y^0 and linking via the binary predicate V the element marked by Y^0 from one class to the element marked by Y^1 from another class with the Y-coordinate greater by one and with the same X-coordinate. The following formula which states that there exists a class with global coordinates $(0, 0)$,

$$\exists x \forall y (x \sim y \to \neg X(y) \wedge \neg Y(y)), \tag{12}$$

guarantees that a class with any pair of coordinates from the range $[0, 2^{2^n} - 1]$, exists. To finish our axiomatization of the grid, it remains to enforce that any pair of global coordinates appears at most once, or, in other words, that any pair of distinct classes have different global coordinates. This is done by means of additional binary predicates $R_0, \ldots, R_{n-1}$ that connect elements from two different classes. The binary predicates define a bit string that indicates the local coordinate on which the bits of X- or Y-coordinates of these classes differ.

$$\forall xyx'y'\big(x \sim y \wedge x' \sim y' \wedge \neg x \sim x' \wedge \bigwedge\nolimits_i (R_i(x,x') \leftrightarrow U_i(y)) \wedge \lambda^{=}(y,y') \\ \rightarrow ((X(y) \leftrightarrow \neg X(y')) \vee (Y(y) \leftrightarrow \neg Y(y')))\big) \quad (13)$$

Consider the conjunction of (2)-(13). It should be clear that each of its models contains, for any $0 \leq p, q < 2^{2^n}$, precisely one class with global coordinates (p, q).

Having established a grid of doubly exponential size, the encoding of any instance of the doubly exponential tiling problem on some tiling system (C, c_0, Hor, Ver) is routine. We simply use the following formulas.

$$\forall x\Big(\bigvee\nolimits_{c \in C} P_c(x) \wedge \bigwedge\nolimits_{c \neq d} \neg(P_c(x) \wedge P_d(x))\Big), \quad (14)$$

$$\bigwedge\nolimits_{c \in C} \forall xy\big(x \sim y \wedge P_c(x) \rightarrow P_c(y)\big), \quad (15)$$

$$\bigwedge\nolimits_{\langle c,d\rangle \notin Hor} \forall xy\big(H(x,y) \wedge P_c(x) \rightarrow \neg P_d(y)\big), \quad (16)$$

$$\bigwedge\nolimits_{\langle c,d\rangle \notin Ver} \forall xy\big(V(x,y) \wedge P_c(x) \rightarrow \neg P_d(y)\big), \quad (17)$$

$$\forall x\,(\forall y(x \sim y \rightarrow (\neg X(y) \wedge \neg Y(y))) \rightarrow P_{c_0}(x))\,. \quad (18)$$

Notice that (18) states that the grid point with coordinates (0,0) is coloured with c_0.

Let Ω be the conjunction of (2)-(18). From any model of Ω, we can read off a $\mathcal{T}$-tiling of size 2^{2^n} for example by inspecting the colours assigned to the elements with local coordinate 0 in each of the $2^{2 \cdot 2^n}$ classes. On the other hand, given any tiling for $\mathcal{T}$, we can construct a finite model of Ω in the obvious way. Thus we see that: (i) if Ω is satisfiable, then $(\mathcal{T}, n)$ has a tiling; (ii) if $(\mathcal{T}, n)$ has a tiling, then Ω is finitely satisfiable. This proves the theorem.

Note that formulas (7) and (13) are in the restricted uniform but not in strongly restricted uniform fragment. The use of $\mathrm{RU}_1(\sim)$ formulas is indeed crucial, since, as we will show later, $\mathrm{SU}_1(\sim)$ has the exponential model property.

5.2 Exponential classes property

In this section we show the following *exponential classes property* of our logics, which then will be used as an important tool in our decidability proofs.

▶ **Lemma 6.** *Let φ be a satisfiable formula in any of the logics $\mathrm{SU}_1(\sim)$, $\mathrm{RU}_1(\sim)$, $\mathrm{U}_1(\sim)$. Then φ has a model in which each equivalence class is bounded exponentially in $\|\varphi\|$.*

Here we show this property for $\mathrm{RU}_1(\sim)$ (which obviously covers also the case of $\mathrm{SU}_1(\sim)$). An extension of the proof covering the case of $\mathrm{U}_1(\sim)$ will be presented in the full version of the paper. The approach we employ is based up to an extent on the approach used in [12] to establish the *small substructures property* for FO^2 (which was further used in that paper to show the exponential classes property for $\mathrm{FO}^2(\sim)$). However, due to considering a richer logic, our proof is technically much more involved.

By Proposition 1, we can restrict attention to normal form formulas. Let us fix a normal form $\mathrm{RU}_1(\sim)$-formula φ of the form given in Equation (1) and a model $\mathfrak{A} \models \varphi$. Let n be the

width of φ, i.e., $n = \max(\{k_i + 1\}_{1 \le i \le m_\exists} \cup \{l_i\}_{1 \le i \le m_\forall})$. Recall that $m_\exists$ is the number of $\forall\exists^*$-conjuncts of φ. To simplify notation, we denote $m := m_\exists$.

In a single step of our construction we consider an equivalence class D in $\mathfrak{A}$, a 1-type α, and the fragment $D_\alpha \subseteq D$ of D consisting of all realizations of α. If $|D_\alpha| \le n$, then we do nothing. Otherwise, we replace D_α by a new fragment bounded polynomially in $\|\varphi\|$, obtaining a new model model $\mathfrak{A}' \models \varphi$. The universe of $\mathfrak{A}'$ consists of $A \setminus D_\alpha$ and a new set D'_α of realizations of α; $\mathfrak{A}' \restriction (A \setminus D'_\alpha) = \mathfrak{A} \restriction (A \setminus D_\alpha)$; and D'_α is formed out of three new disjoint sets D^0_α, D^1_α, D^2_α such that $D^i_\alpha = \{a^i_1, \dots, a^i_{mn}\}$ for $i = 1, 2$, and $D^0_\alpha = \{a^0_1, \dots, a^0_{(m+1)^3 n^2}\}$. For a set $B \subseteq A$, we call $B \setminus D_\alpha$ its *external fragment* and $B \cap D_\alpha$ its *internal* fragment. We also speak about external and internal fragments of witness structures and use analogous terminology for $\mathfrak{A}'$. The construction of $\mathfrak{A}'$ is divided into several stages.

Labelling of subsets. We take a set of labels $L_1, \dots, L_m$. For each subset B of $A \setminus D_\alpha$ such that $1 \le |B| < n$, for each $a \in D_\alpha$, for each $1 \le i \le m$: if B forms the external fragment of the live part of a witness structure for a and $\varphi^\exists_i$ in $\mathfrak{A}$, then label B with L_i. Note that some subsets B may be labelled by several L_is, and some may have no labels.

Let L^* be a fresh label. For every $b \in A \setminus D_\alpha$ and every $1 \le i \le m$, choose a witness structure for b and $\varphi^\exists_i$ in $\mathfrak{A}$. If the internal fragment of the live part of this witness structure is not empty, then label the set of elements of this live part with L^*. Later on, we will take care of replicating such a witness structure for b in $\mathfrak{A}'$.

Special subsets. We collect some subsets of $A \setminus D_\alpha$ into a *set* Θ *of special subsets*. This set will be sufficiently rich to provide the external fragments of the live parts of witness structures for any element in D'_α. For each label L_i, $1 \le i \le m$, if there are at most m subsets of $A \setminus D_\alpha$ labelled by L_i, then make all of them members of Θ; call such a label *rare*. Otherwise, choose $m + 1$ such subsets and make them members of Θ. Note that $|\Theta| \le (m + 1)m$, and thus it is bounded polynomially in $\|\varphi\|$.

Witnesses for Θ. Now, for each subset $B \in \Theta$, we replicate in $\mathfrak{A}'$ those witness structures from $\mathfrak{A}$ whose live parts are labelled by L^* and their external fragments equal B. Assume $B = \{b_1, \dots, b_k\}$. For each $\{a_1, \dots, a_l\} \subseteq D_\alpha$, $l \ge 1$, such that $\{b_1, \dots, b_k, a_1, \dots, a_l\}$ is labelled by L^*, take fresh elements $a'_1, \dots, a'_l$ from D^0_α and set $\mathrm{tb}_{\mathfrak{A}'}(b_1, \dots, b_k, a'_1, \dots, a'_l) := \mathrm{tb}_{\mathfrak{A}}(b_1, \dots, b_k, a_1, \dots, a_l)$. We simultanously begin defining a *pattern function* $f : D'_\alpha \to D_\alpha$ by setting $f(a'_i) := a_i$ for $1 \le i \le l$. Let us estimate the number of elements in D^0_α required for this step: There are at most $(m + 1)m$ subsets B in Θ, each of them of size smaller than n. In Step *Labelling of subsets*, each element of such B could produce at most m witness structures labelled with L^*, and each such structure has less than n elements in D_α. Thus we need at most $(m + 1)m(n - 1)m(n - 1)$ elements, and we indeed have that many, as we declared D^0_α to have $(m + 1)^3 n^2$ elements.

For all elements a' of D^0_α not used in the above step, as well as for elements a' from $D^1_\alpha \cup D^2_\alpha$, choose an arbitrary element $a \in D_\alpha$ and set $f(a') := a$.

Witnesses for elements of D^0_α. Let $a' \in D^0_\alpha$. If there is a subset in Θ such that a' was used to replicate a witness structure labelled by L^* in stage *Witnesses for* Θ, then call this set $B^*_{a'}$ (by our construction, there is at most one such set). We then continue without assuming that a' was necessarily used for replicating a set labelled L^*. Let $a = f(a')$ be the *pattern element* for a'. For each $1 \le j \le m$ such that L_j is rare, find a witness structure $\mathfrak{B}_j$ for a and $\varphi^\exists_j$ in $\mathfrak{A}$. Assume that $B_j = \{a, a_1 \dots, a_k, a_{k+1}, \dots, a_s, b_1, \dots, b_l, b_{l+1}, \dots, b_t\}$,

with $a_i \in D_\alpha$ for $1 \leq i \leq s$ and $b_i \in A \setminus D_\alpha$ for $1 \leq i \leq t$, and that the live part of $\mathfrak{B}_j$ is $\bar{B}_j = \{a, a_1 \ldots, a_k, b_1, \ldots, b_l\}$. It may happen that $l = 0$, which means that the live part of the witness structure is contained in D_α. Otherwise, the set $\{b_1, \ldots, b_l\}$ is labelled by L_j (and possibly some other labels) and is a member of Θ. We set $\mathrm{tb}_{\mathfrak{A}'}(a, a^1_{(j-1)n+1}, \ldots, a^1_{(j-1)n+k}, b_1, \ldots, b_l) := \mathrm{tb}_{\mathfrak{A}}(a, a_1, \ldots, a_k, b_1, \ldots, b_l)$. Note that this guarantees that the structure defined on the set $\{a', a^1_{(j-1)n+1}, \ldots, a^1_{(j-1)n+k}, a^1_{(j-1)n+k+1}, \ldots, a^1_{(j-1)n+s}, b_1, \ldots, b_l, b_{l+1}, \ldots, b_t\}$ will be a witness structure for a' and $\varphi^\exists_j$ in $\mathfrak{A}'$. Let us here comment one subtlety: if $k = 0$, then it is possible that $\mathrm{tb}_{\mathfrak{A}'}(a, b_1, \ldots, b_l)$ was defined before (either in this stage for some different j, or if $B^*_{a'} = \{b_1, \ldots, b_l\}$, in the step *Witnesses for* Θ). Note, however, that there is no danger of conflict here, since this earlier definition must agree with the new one.

For each $1 \leq j \leq m$ such that L_j is not rare, let $\{b_1, \ldots, b_l\}$ be a subset from Θ labelled by L_j, different from $B^*_{a'}$, and not yet used for a' for any other j (note that such a subset exists, since there are $m + 1$ subsets labelled by L_j in Θ, and at most $m - 1$ of them can be used as the external parts of witness structures for a' for other js). Let $a \in D_\alpha$, and let $\mathfrak{B}_j$ be a witness structure for a and $\varphi^\exists_j$ in $\mathfrak{A}$ in which the external fragment of the live part is formed by $\{b_1, \ldots, b_l\}$. Such a and $\mathfrak{B}_j$ exist due to the construction of Θ. Assume that $B_j = \{a, a_1 \ldots, a_k, a_{k+1}, \ldots, a_s, b_1, \ldots, b_l, b_{l+1}, \ldots, b_t\}$, with $a_i \in D_\alpha$ for $1 \leq i \leq s$, $b_i \in A \setminus D_\alpha$ for $1 \leq i \leq t$. Assume the live part of B_j is $\bar{B}_j = \{a, a_1 \ldots, a_k, b_1, \ldots, b_l\}$. We set $\mathrm{tb}_{\mathfrak{A}'}(a', a^1_{(j-1)n+1}, \ldots, a^1_{(j-1)n+k}, b_1, \ldots, b_l) := \mathrm{tb}_{\mathfrak{A}}(a, a_1, \ldots, a_k, b_1, \ldots, b_l)$. This guarantees that the structure defined on the set $\{a', a^1_{(j-1)n+1}, \ldots, a^1_{(j-1)n+k}, a^1_{(j-1)n+k+1}, \ldots, a^1_{(j-1)n+s}, b_1, \ldots, b_l, b_{l+1}, \ldots, b_t\}$ will be a witness structure for a' and $\varphi^\exists_j$ in $\mathfrak{A}'$.

Witnesses for D^1_α and D^2_α. Witness structures for $a' \in D^1_\alpha \cup D^2_\alpha$ are provided in a similar way to that described for elements of D^0_α, but if $a' \in D^1_\alpha$ ($a' \in D^2_\alpha$), then we take elements a'_i from D^2_α (D^0_α). This cyclic scheme guarantees that the procedure avoids conflicts.

Witnesses for subsets of $A \setminus D'_\alpha$ not belonging to Θ. Let $B = \{b_1, \ldots, b_l\}$ be a subset of $A \setminus D_\alpha$, $l \leq n$, not belonging to Θ. Note that no table with external part B has yet been defined. The number of subsets labelled by L^* whose external part equals B is bounded above by mn. Since the internal part of each of them has less than n elements, we can replicate them without conflicts using mn^2 elements of D^0_α.

Completion. Let $\{a'_1, \ldots, a'_k, b_1, \ldots, b_l\} \subseteq A'$ be such that $a'_i \in D'_\alpha$ for $1 \leq i \leq k$, $b_i \in A \setminus D_\alpha$ for $1 \leq i \leq l$, $k + l \leq n$, and such that $\mathrm{tb}_{\mathfrak{A}'}(a'_1, \ldots, a'_k, b_1, \ldots, b_l)$ has not been defined yet. Take any pairwise distinct elements $a_1, \ldots, a_k$ from D_α (such elements exists, since $k \leq n$ and $|D_\alpha| \geq n$) and set $\mathrm{tb}_{\mathfrak{A}'}(a'_1, \ldots, a'_k, b_1, \ldots, b_l) := \mathrm{tb}_{\mathfrak{A}}(a_1, \ldots, a_k, b_1, \ldots, b_l)$.

This finishes the construction for replacing D_α by a small set D'_α. We now argue that the obtained model $\mathfrak{A}'$ is indeed a model of φ.

▶ **Claim 7.** $\mathfrak{A}' \models \varphi$.

Proof. Let us see first that all elements have the required witness structures. Let $b \in A'$ and $1 \leq i \leq m$. If $b \in D'_\alpha$, then an appropriate witness structure for b and $\varphi^\exists_j$ was constructed in the step *Witnesses for* D^0_α or *Witnesses for* D^1_α *and* D^2_α. Assume that $b \in A' \setminus D'_\alpha$ ($= A \setminus D_\alpha$). Either b has a witness structure for $\varphi^\exists_i$ in $\mathfrak{A}\restriction A \setminus D_\alpha$ and this structure is inherited into $\mathfrak{A}'$, or, in the step *Labelling* we labelled with L^* at least one witness structure $\mathfrak{B}$ for b and $\varphi^\exists_j$ in $\mathfrak{A}$. If the external fragment of the live part $\bar{B}$ of this structure is in Θ, then the live part of the corresponding witness structure in $\mathfrak{A}'$ was defined in step *Witnesses for* Θ. Otherwise it was

defined in step *Witness for subsets of* $A \setminus D'_\alpha$ *not belonging to* Θ. The necessary members of the witness structure which are not live can easily be found.

Consider now a conjunct of φ of the form $\forall x_1, \ldots, x_u \varphi_j^\forall$ and a tuple of not necessarily distinct elements $d'_1, \ldots, d'_u \in A'$. We want to see that $\mathfrak{A}' \models \varphi_j^\forall[d'_1, \ldots, d'_u]$. Let us enumerate the elements of $\{d'_1, \ldots, d'_u\}$ by $a'_1, \ldots, a'_k, a'_{k+1}, \ldots, a'_s, b_1, \ldots, b_l, b_{l+1}, \ldots, b_t$, where $a'_i \in D'_\alpha$ for $1 \leq i \leq s$, $b_i \in A \setminus D'_\alpha$ for $1 \leq i \leq t$, and $\mathrm{live}(\varphi_j^\forall[d'_1, \ldots, d'_u]) = \{a'_1, \ldots, a'_k, b_1, \ldots, b_l\}$. By our construction, $\mathrm{tb}_{\mathfrak{A}'}(a'_1, \ldots, a'_k, b_1, \ldots, b_l) = \mathrm{tb}_{\mathfrak{A}}(a_1, \ldots, a_k, b_1, \ldots, b_l)$ for some $a_1, \ldots, a_k \in D_\alpha$ (if $k = 0$, then simply $\mathrm{tb}_{\mathfrak{A}'}(b_1, \ldots, b_l) = \mathrm{tb}_{\mathfrak{A}}(b_1, \ldots, b_l)$; otherwise the table was set either in one of *Witnesses for ...* -stages or in the *Completion* stage). Take now any pairwise distinct elements $a_{k+1}, \ldots, a_s \in D_\alpha$ different from $a_1, \ldots, a_k$ (this is possible since $s \leq n$ and there are at least n elements in D_α) and observe that the equivalence relations among $a_1, \ldots, a_s, b_1, \ldots, b_t$ are isomorphic to those among $a'_1, \ldots, a'_s, b_1, \ldots, b_t$. This guarantees that $\mathfrak{A}' \models \varphi_j^\forall[d'_1, \ldots, d'_u]$. ◀

Now, to find a small replacement of a whole class, we apply the described construction iteratively to all D_α, where α is a 1-type realized in this class. Let $D_1, D_2, \ldots$ be a (possibly infinite) sequence of the classes in a $\mathfrak{A}$ (we can assume that $\mathfrak{A}$ is countable due to the Löwenheim-Skolem property), let $\mathfrak{A}_0 = \mathfrak{A}$, and let $\mathfrak{A}_{j+1}$ be the structure $\mathfrak{A}_j$ modified by replacing class D_{j+1} by the small replacement class D'_{j+1} as described above. The obtained natural limit structure is the desired model with exponentially bounded classes.

5.3 Exponential model property for $\mathrm{SU}_1(\sim)$

Recall that in Section 5.1 we proved a 2-NExpTime lower bound for $\mathrm{RU}_1(\sim)$. Here we show that $\mathrm{SU}_1(\sim)$ is easier. To understand why the complexity drop is possible, consider the conjunct (6) of the formula Ω from Section 5.1. When we look for a witness structure for an element a satisfying X^0 and this conjunct, we have to find an appropriate element b (i.e., an element with the same local coordinate as a, satisfying X^1, connected to a by H), but additionally, due to the conjunct (7), we must take into account the 1-types of elements from the classes of a and b that *do not belong to the witness structure.* The restrictions of $\mathrm{SU}_1(\sim)$ would not enable this. Indeed we can now prove the following theorem.

▶ **Theorem 8.** *The satisfiability problem for* $\mathrm{SU}_1(\sim)$ *is* NExpTime*-complete.*

To prove this theorem, we establish the exponential model propery. Thus checking if a given formula is satisfiable can be done by nondeterministically guessing a structure of exponentially bounded size and verifying that it is indeed a model. Such a model checking task (for normal form formulas) can be done in polynomial time in a straighforward way. The matching lower bound follows from the NExpTime-hardness of FO^2.

▶ **Lemma 9.** *Every satisfiable* $\mathrm{SU}_1(\sim)$ *formula* φ *has a finite model of size bounded exponentially in* $\|\varphi\|$*.*

We next prove this lemma. For the rest of this section, fix a normal form formula φ of $\mathrm{SU}_1(\sim)$ and a model $\mathfrak{A} \models \varphi$. Due to Lemma 6, we may assume that the equivalence classes of $\mathfrak{A}$ are bounded exponentially in $\|\varphi\|$. As previously, let n be the width of φ and m the number of $\forall\exists^*$-conjuncts of φ. We construct a small model $\mathfrak{A}' \models \varphi$ in several stages.

Court. If a k-multitype, for $1 \leq k \leq n$, is realized in less than n classes of $\mathfrak{A}$, then call all these classes *royal*. If a k-multitype, for $1 \leq k \leq n$, is realized only in royal classes, then call this multitype *royal*. Note that it is possible that a multitype realized in more than n

classes is royal. Let K be the union of all royal classes of $\mathfrak{A}$. For each $a \in K$ and for each conjunct $\varphi_i^\exists$ of φ, find a witness structure $\mathfrak{C}_{a,i}$ for a and $\varphi_i^\exists$ in $\mathfrak{A}$. Let C be the union of K and all the classes containing some element from some $C_{a,i}$. Note that the size of C is bounded exponentially in $\|\varphi\|$. $\mathfrak{C}$ is called the *court* of $\mathfrak{A}$.

Universe. For all $1 \leq k \leq n$, for each non-royal k-multitype θ realized in a class of $\mathfrak{A}$, *appoint one* such a class D_θ. We build a new model $\mathfrak{A}' \models \varphi$ whose universe is $C \cup \bigcup D^s_{\theta,u,w}$, where each $D^s_{\theta,u,w}$ is a fresh set and the union is taken over all non-royal k-multitypes θ realized in a class of $\mathfrak{A}$ ($1 \leq k \leq n$), $0 \leq s \leq 2$, $1 \leq u \leq n$, $1 \leq w \leq m$. For $i = 0, 1, 2$, let D^i be the union of all $D^s_{\theta,u,w}$ with $s = i$. We make $\mathfrak{A}' \restriction C$ isomorphic to $\mathfrak{A} \restriction C$, and $\mathfrak{A}' \restriction (K \cup D^s_{\theta,u,w})$ isomorphic to $\mathfrak{A} \restriction (K \cup D_\theta)$, for all relevant θ, s, u, w. For each $D^s_{\theta,u,w}$, let $g^s_{\theta,u,w} : D^s_{\theta,u,w} \to D_\theta$ be an isomorphism. Also, for a class D in $\mathfrak{C}$, let $g_D : D \to D$ be the identity function. Define $g : A' \to A$ to be $g := \bigcup_{D \in C/\sim} g_D \cup \bigcup g^s_{\theta,u,w}$, where the second union is taken over all relevant θ, s, u, w. We call g the *pattern function*. It will return, for each element of $\mathfrak{A}'$, a 'similar' element in $\mathfrak{A}$. At this stage the structure of $\mathfrak{A}'$ is defined on $\mathfrak{C}$, on each equivalence class, and on each union of a non-royal class with K. The size of $\mathfrak{A}'$ is exponentially bounded in $\|\varphi\|$, as required.

Witnesses. Let $a' \in A' \setminus K$. Let a be the *pattern element* for a', $a := g(a')$. For each $1 \leq j \leq m$, find a witness structure $\mathfrak{B}_{a,j}$ for a and $\varphi_j^\exists$ in $\mathfrak{A}$. We want to define a similar witness structure for a' in $\mathfrak{A}'$. We consider explicitly the case in which $a' \in D^0$. Assume that the class of a' is $D^0_{\theta,u,w}$. Then $a \in D_\theta$. Let $T_0, T_1, \ldots, T_s, T_{s+1}, \ldots, T_t$ be the division of $\mathfrak{B}_{a,j}$ into classes such that $a \in T_0 \subseteq D_\theta$, the sets $T_1, \ldots, T_s$ (possibly $s = 0$) are fragments of non-royal classes of $\mathfrak{A}$ and $T_{s+1}, \ldots, T_t$ (possibly $t - s = 0$) are fragments of royal classes of $\mathfrak{A}$. Let θ_i be the multitype of T_i, for $1 \leq i \leq t$. We define a function $h : B_{a,j} \to A'$ whose image is supposed to form a witness structure for a' in $\mathfrak{A}'$. For $i = 1, \ldots, s$, let T'_i be a set of multitype θ_i from $D^1_{\theta_i,i,j}$ (recall that $i \leq s \leq n$), and let $h_i : T_i \to T'_i$ be a bijection preserving 1-types. We set

$$h(b) := \begin{cases} b' \text{ such that } g(b') = b & \text{if } b \in T_0; \\ h_i(b) & \text{if } b \in T_i \text{ for } 1 \leq i \leq s; \\ b & \text{if } b \in T_i \text{ for } i > s. \end{cases}$$

Let $b_1, \ldots, b_k$ be an enumeration of the elements of $B_{a,j}$. If $s = 0$, i.e., all elements of $B_{a,j} \setminus \{a\}$ are in royal classes, then $\mathfrak{A}' \restriction \{h(b_1), \ldots, h(b_k)\}$ already forms a witness structure for a and $\varphi_j^\exists$. Otherwise we set $\mathrm{tb}_{\mathfrak{A}'}(h(b_1), \ldots, h(b_k)) := \mathrm{tb}_{\mathfrak{A}}(b_1, \ldots, b_k)$.

If $a' \in D^1$, then we proceed similarly, but use elements of D^2 instead of elements of D^1; if $a' \in D^2$ or $a' \in K \setminus C$, then we use elements of D^0 instead of elements of D^1. This circular witnessing scheme, inspired by the one from [7], together with the strategy of using an appropriate number of copies of classes, guarantees that for each subset $B \subseteq A'$, the table for some enumeration of its elements is defined at most once.

Completion. Let $a'_1, \ldots, a'_k$, $1 \leq k \leq n$, be a tuple of elements from $\mathfrak{A}'$ whose table is not yet defined. Let $T'_1, \ldots, T'_s, T'_{s+1}, \ldots, T'_t$ be a partition of $\{a'_1, \ldots, a'_k\}$ into classes such that the sets $T'_1, \ldots, T'_s$ are fragments of non-royal classes of $\mathfrak{A}'$ (this time $s > 0$, since otherwise all elements of the tuple would belong to K, and thus their table would have been already defined) and $T'_{s+1}, \ldots, T'_t$ (possibly $t - s = 0$) are fragments of royal classes of $\mathfrak{A}'$. Assume that the multitype of T'_i is θ_i for $1 \leq i \leq t$. Since $s \leq n$, and as the multitypes of

$T'_1, \dots, T'_s$ are non-royal, we can find in distinct classes of $\mathfrak{A}$ subsets $T_1, \dots, T_s$ such that the multitype of T_i in $\mathfrak{A}$ equals the multitype of T'_i in $\mathfrak{A}'$. For $1 \leq i \leq t$, let $h'_i : T'_i \to T_i$ be a bijection preserving 1-types (for $i > s$ it can be just the identity). Let $h' = \bigcup_{1 \leq i \leq t} h'_i$. Set $\mathrm{tb}_{\mathfrak{A}'}(a'_1, \dots, a'_k) := \mathrm{tb}_{\mathfrak{A}}(h'(a'_1), \dots, h'(a'_k))$. This finishes the construction of $\mathfrak{A}'$.

We provided witness structures for $\forall\exists^*$-conjuncts of φ for elements from K in step *Court*, and for elements from $A' \setminus K$ in step *Witnesses*. All universal conjuncts of φ are satisfied since for any tuple of elements from A', its table is defined precisely as a table in $\mathfrak{A}$ for some elements with the same 1-types and isomorphic equivalence connections. Thus

▶ **Claim 10.** $\mathfrak{A}' \models \varphi$.

5.4 Doubly exponential model property for $\mathrm{RU}_1(\sim)$ and $\mathrm{U}_1(\sim)$

The following result completes our discourse on complexity.

▶ **Theorem 11.** *The satisfiability problems for* $\mathrm{RU}_1(\sim)$ *and* $\mathrm{U}_1(\sim)$ *are 2-*NExpTime*-complete.*

The lower bound for $\mathrm{RU}_1(\sim)$ (and thus $\mathrm{U}_1(\sim)$) was shown in Theorem 5. The matching upper bound follows from the following small model property.

▶ **Lemma 12.** *Every satisfiable formula* φ *of* $\mathrm{RU}_1(\sim)$ *or* $\mathrm{U}_1(\sim)$ *has a finite model of size bounded doubly exponentially in* $\|\varphi\|$.

A proof of this lemma will appear in the full version of this paper. Here we only remark that instead of working with multitypes of small subsets of classes, as we did in Section 5.3 in the case of $\mathrm{SU}_1(\sim)$, this time we consider multitypes of whole classes. Due to the exponential bound on the size of classes, the number of possible multitypes of classes in a model is bounded doubly exponentially in $\|\varphi\|$. The proof consists of reproducing an appropriate number of realizations of every multitype in the new small model. The basic structure of our small model construction is similar to that in Section 5.3.

5.5 Limits of decidability

We consider two natural generalisations of the weakest of our logics $\mathrm{SU}_1(\sim)$ and show their undecidability. Let us denote by $\mathrm{SU}_1(\sim_1, \sim_2)$ the extension of $\mathrm{SU}_1(\sim)$ in which there are two distinguished binary predicates which must be interpreted as equivalences (and can be used freely), rather than just one. Let $\mathrm{SU}_1(tr)$ be the extension of SU_1 in which a distinguished binary symbol tr must be interpreted as an arbitrary transitive relation (and can be used freely), rather than as an equivalence relation. Note that $\mathrm{SU}_1(tr)$ is an extension of $\mathrm{SU}_1(\sim)$ since reflexivity and symmetry of tr can be enforced in SU_1 in a straighforward way.

▶ **Theorem 13.** *The satisfiability and finite satisfiability problems for* $\mathrm{SU}_1(\sim_1, \sim_2)$ *and* $\mathrm{SU}_1(tr)$ *are undecidable.*

Proof. Recall the tiling systems from Section 5.1. We define the standard infinite grid as $\mathfrak{G}_{\mathbb{N}} = (\mathbb{N} \times \mathbb{N}, H, V)$, $H = \{((p,q),(p',q)) : p' - p = 1\}$, $V = \{((p,q),(p,q')) : q' - q = 1\}$. The standard grid $\mathfrak{G}^*_m$ on a finite torus is defined as $\mathfrak{G}_m$ from Section 5.1, with additional horizontal H-edges from the last to the first column, and additional vertical V-edges from the last to the first row. It is well known that the problem of checking if a tiling system $\mathcal{T}$ tiles the standard infinite grid $\mathfrak{G}_{\mathbb{N}}$, and the problem of checking if it tiles a toroidal grid $\mathfrak{G}^*_m$

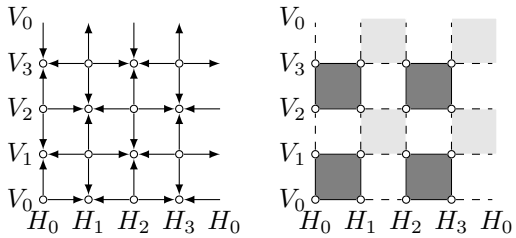


Figure 1 Grid structures $\hat{\mathfrak{G}}_{\mathbb{N}}$ for $\mathrm{SU}_1(tr)$ and $\check{\mathfrak{G}}_{\mathbb{N}}$ for $\mathrm{SU}_1(\sim_1, \sim_2)$. Arrows indicate tr connections, solid edges represent $\sim_1$, dashed edges represent $\sim_2$, equivalence classes of $\sim_1$ and $\sim_2$ are indicated by, respectively, dark and light shadings.

for some $m \in \mathbb{N}$, are undecidable. We can encode these problems in $\mathrm{SU}_1(tr)$ and $\mathrm{SU}_1(\sim_1, \sim_2)$ quite easily. We concentrate on the proof for $\mathrm{SU}_1(tr)$. The proof for $\mathrm{SU}_1(\sim_1, \sim_2)$ is similar.

For a given tiling system $\mathcal{T}$, we construct an $\mathrm{SU}_1(tr)$ formula Θ. Our intended grid expansion $\hat{\mathfrak{G}}_{\mathbb{N}}$ is illustrated in Fig. 1. It interprets auxiliary unary symbols H_i, V_i, for $0 \leq i, j \leq 3$, and, obviously, the transitive symbol tr. It is crucial that $\hat{\mathfrak{G}}_{\mathbb{N}}$ avoids binary connections between points which are distant from each other.

We capture properties of horizontally and vertically neighbouring elements by formulas $\lambda_H(x, y)$ and $\lambda_V(x, y)$,

$$\lambda_H(x, y) \equiv \bigvee_{0 \leq i,j \leq 3} \lambda_H^{i,j}(x, y), \tag{19}$$

where $\lambda_H^{i,j} \equiv H_i(x) \wedge H_{i+1}(y) \wedge V_j(x) \wedge V_j(y) \wedge tr^{i+j}(x, y)$; here $i+1$ is taken modulo 4, and $tr^k(x, y)$ denotes $tr(x, y)$ for even k, and $tr(y, x)$ for odd k. $\lambda_V(x, y)$ is defined analogously. Grid coordinate points are appropriately completed:

$$\forall x(\exists y \lambda_H(x, y) \wedge \exists y \lambda_V(x, y)), \tag{20}$$

$$\forall xyzt(\lambda_H(y, z) \wedge \lambda_V(y, x) \wedge \lambda_V(z, t) \to \lambda_H(x, t)). \tag{21}$$

Finally, we encode an instance of the tiling problem $\mathcal{T} = (C, c_0, Hor, Ver)$, similarly to the way we did it in Section 5.1.

$$\exists x(H_0(x) \wedge V_0(x) \wedge P_{c_0}(x)), \tag{22}$$

$$\forall x\Big(\bigvee_{c \in C} P_c(x) \wedge \bigwedge_{c \neq d} \neg(P_c(x) \wedge P_d(x))\Big), \tag{23}$$

$$\bigwedge\nolimits_{\langle c,d \rangle \notin Hor} \forall xy\big(\lambda_H(x, y) \wedge P_c(x) \to \neg P_d(y)\big), \tag{24}$$

$$\bigwedge\nolimits_{\langle c,d \rangle \notin Ver} \forall xy\big(\lambda_V(x, y) \wedge P_c(x) \to \neg P_d(y)\big). \tag{25}$$

Let Θ be the conjunction of (20)-(25). We claim that Θ is satisfiable iff $\mathcal{T}$ tiles $\mathfrak{G}_{\mathbb{N}}$, and that Θ is finitely satisfiable iff $\mathcal{T}$ tiles $\mathfrak{G}_m^*$ for some $m \in \mathbb{N}$. We sketch the argument for the first part of the claim. Assume that $\mathcal{T}$ tiles $\mathfrak{G}_{\mathbb{N}}$. Take a tiling $f : G_{\mathbb{N}} \to C$, and consider the expansion of $\hat{\mathfrak{G}}_{\mathbb{N}}$ which satisfies $P_{f(i,j)}[i, j]$ for every $i, j \in \mathbb{N}$. It is readily verified that it is a model of Θ (here it is important that in our arrangement of the tr-arrows, the transitivity of tr does not enforce connections between distant points). In the opposite direction, if Θ has a model $\mathfrak{M}$, then using (20)-(22) we can define a homomorphism $F : \mathfrak{G}_{\mathbb{N}} \to \mathfrak{M}$ mapping $\langle 0, 0 \rangle$ to an element that satisfies P_{c_0}. Further, using (22)-(25), we can define a tiling f of $\mathfrak{G}_{\mathbb{N}}$ by

setting $f(i,j) = c$ for the unique c such that $\mathfrak{M} \models P_c[F(i,j)]$. We skip here the (routine) argument for the case of finite satisfiability. This finishes the proof for $\mathrm{SU}_1(tr)$.

The case of $\mathrm{SU}_1(\sim_1, \sim_2)$ is treated analogously. The only changes are that we use $\check{\mathfrak{G}}_{\mathbb{N}}$ from Fig. 1 instead of $\hat{\mathfrak{G}}_{\mathbb{N}}$, and modify appropriately the definitions of $\lambda_H(x,y)$ and $\lambda_V(x,y)$. ◀

The above undecidability results contrast with the fact that FO^2 remains decidable when extended by two equivalence relations [12, 13] or one transitive relation [23]. It should be emphasised, however, that our undecidability proofs exploit the free, non-uniform use of the special relation symbols (as in conjunct (21)), rather than transitivity of the relations corresponding to the symbols. (Actually, the presented arguments work in a natural way if we do not require tr to be interpreted as a transitive relation.) It is likely that the decidability can be regained if we require the special symbols to obey the regular uniformity constraints. We leave this, however, for future work.

Acknowledgements. The first author was partially supported by the Polish National Science Centre grant DEC-2013/09/B/ST6/01535. The second author was supported by Jenny and Antti Wihuri Foundation.

References

1 H. Andréka, J. van Benthem, and I. Németi. Modal languages and bounded fragments of predicate logic. *Journal of Philosophical Logic*, 27:217–274, 1998.
2 V. Bárány, B. ten Cate, and L. Segoufin. Guarded negation. In *ICALP 2011, Proceedings Part II*, pages 356–367, 2011.
3 S. Benaim, M. Benedikt, W. Charatonik, E. nski, R. Lenhardt, F. Mazowiecki, and J. Worrell. Complexity of two-variable logic on finite trees. In *ICALP 2013, Proceedings Part II*, pages 74–88, 2013.
4 M. Bojanczyk, C. David, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on data words. *ACM Transactions on Computational Logic*, 12(4):27, 2011.
5 M. Bojanczyk, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on data trees and XML reasoning. *Journal of the ACM*, 56(3), 2009.
6 H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer, 1885.
7 E. Grädel, P. Kolaitis, and M. Vardi. On the decision problem for two-variable first-order logic. *Bulletin of Symbolic Logic*, 3(1):53–69, 1997.
8 E. Grädel, M. Otto, and E. Rosen. Two-variable logic with counting is decidable. In *LICS 1997, Proceedings*, pages 306–317, 1997.
9 L. Hella and A. Kuusisto. One-dimensional fragment of first-order logic. In *AIML 2014, Proceedings*, pages 274–293, 2014.
10 E. Kieronski and A. Kuusisto. Complexity and expressivity of uniform one-dimensional fragment with equality. In *MFCS 2014, Proceedings Part I*, pages 365–376, 2014.
11 E. Kieronski, J. Michaliszyn, I. Pratt-Hartmann, and L. Tendera. Two-variable first-order logic with equivalence closure. *SIAM Journal on Computing*, 43(3):1012–1063, 2014.
12 E. Kieronski and M. Otto. Small substructures and decidability issues for first-order logic with two variables. *Journal of Symbolic Logic*, 77:729–765, 2012.
13 E. Kieronski and L. Tendera. On finite satisfiability of two-variable first-order logic with equivalence relations. In *LICS 2009, Proceedings*, pages 123–132, 2009.
14 B. ten Cate L. Segoufin. Unary negation. *Logical Methods in Computer Science*, 9(3), 2013.
15 L. Libkin. *Elements of Finite Model Theory*. Springer, 2004.
16 A. Montanari and P. Sala. Adding an equivalence relation to the interval logic AB$\bar{\mathrm{B}}$: complexity and expressiveness. In *LICS 2013, Proceedings*, pages 193–202, 2013.

17 M. Mortimer. On languages with two variables. *Zeitschr. f. math. Logik u. Grundlagen d. Math.*, 21:135–140, 1975.

18 L. Pacholski, W. Szwast, and L. Tendera. Complexity of two-variable logic with counting. In *LICS 1997, Proceedings*, pages 318–327, 1997.

19 C.H. Papadimitriou. *Computational Complexity*. Addison Wesley Longman, 1994.

20 I. Pratt-Hartmann. Complexity of the two-variable fragment with counting quantifiers. *Journal of Logic, Language and Information*, 14(3):369–395, 2005.

21 I. Pratt-Hartmann. Logics with counting and equivalence. In *CSL-LICS 2014, Proceedings*, page 76, 2014.

22 D. Scott. A decision method for validity of sentences in two variables. *Journal of Symbolic Logic*, 27:477, 1962.

23 W. Szwast and L. Tendera. FO^2 with one transitive relation is decidable. In *STACS 2013, Proceedings*, pages 317–328, 2013.

Finite-Degree Predicates and Two-Variable First-Order Logic

Charles Paperman

University of Warsaw, Poland

Abstract

We consider two-variable first-order logic on finite words with a fixed number of quantifier alternations. We show that all languages with a neutral letter definable using the order and finite-degree predicates are also definable with the order predicate only. From this result we derive the separation of the alternation hierarchy of two-variable logic on this signature.

1998 ACM Subject Classification F.4.3 Formal Languages

Keywords and phrases First order logic, automata theory, semigroup, modular predicates

Digital Object Identifier 10.4230/LIPIcs.CSL.2015.616

1 Introduction

Finite model theory and the lower classes of circuit complexity are intricately interwoven. In the context of circuit complexity, logics are considered over finite words with *arbitrary numerical predicates*. Intuitively, we allow the use of any predicate that only depends on the size of the word. A first result from Immerman [6] provides an equivalence between languages definable by first-order logic enriched with arbitrary numerical predicates on the one hand, and languages computable by families of circuits of constant depth and polynomial size on the other. Since then, several meaningful circuit complexity classes have been shown to be equivalent to logical fragments [1, 8]. It is therefore possible to obtain deep and interesting inexpressibility results by using circuits lower bounds.

For instance, by using a famous lower bound for the *parity* language [5], Barrington, Compton, Straubing and Thérien [1] showed that the regular languages definable in first-order logic with arbitrary numerical predicates are definable with only the *regular predicates*. Relying on an algebraic description of first-order logic with regular predicates, it is possible to decide the definability of a regular language in this logic.

Conversely, it is tempting to use finite model theory methods to compute circuit lower bounds. This approach has achieved relative success for *uniform* versions of circuit complexity classes. For instance, Roy and Straubing provide a separation result for the long-standing question of the separation of **ACC** from $\mathbf{NC^1}$ in a highly uniform setting [18]. In these settings, this uniformity condition has two different interpretations:

1. In the circuit framework, it is a restriction on the complexity of the wiring of the gates.
2. In the logical framework, it is a restriction on the class of numerical predicates considered in the fragment.

In order to deal with the combinatorics of arbitrary numerical predicates, the languages with a *neutral letter* have been introduced in [2]. Formally, a language L has a neutral letter c if for any pair of words u, v, we have $ucv \in L$ if, and only if, $uv \in L$. Less formally, this letter c can be added or removed anywhere in a word without changing its membership to L. The underlying idea was that numerical predicates would be essentially useless in the presence of a neutral letter. This was made formal through the *Crane Beach conjecture*:

24th EACSL Annual Conference on Computer Science Logic (CSL 2015).
Editor: Stephan Kreutzer; pp. 616–630

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Every language with a neutral letter definable in first-order logic with arbitrary numerical predicates is definable in first-order logic with the linear order only.

Furthermore, some of the most interesting languages, such as the *parity* language, possess a neutral letter. Unfortunately, this conjecture has been disproved in the article [2] in the context of first-order logic, as long as the Bit predicate is in the signature. This result prevents the use of this approach to obtain circuit lower bounds for more expressive classes. However, for fragments of first-order logic the Crane Beach conjecture is still of interest. For instance, the Crane Beach conjecture holds for the fragment without quantifier alternation [2].

Turning to other fragments, two-variable first-order logic is a robust and well-studied class that offers a wide range of long-standing and intriguing open questions. It is not know whether the Crane Beach conjecture holds for this fragment. This question is related to a long-standing open linear lower bound for the addition function, since two-variable logic is equivalent to linear circuits of $\mathbf{AC}^0$ [8]. Therefore, if the Crane Beach conjecture holds, then the addition function is not computable by a constant-depth linear-size circuit family. This result would improve on a known lower bound for addition that states that addition is not computable by circuits of constant depth with a linear number of wires [3]. We remark that lower bound for addition has been discussed and informally mentioned several times [16, 4, 9, 8] and formally stated in the article [7, Open problem 23].

In this paper, we focus on the case of two-variable logic, which is poorly understood in this context. We first prove that languages with a neutral letter definable in two-variable logic with arbitrary numerical predicates can be defined allowing only the linear order and the following predicates:

1. The class $\mathcal{F}$ of finite-degree predicates, that is, binary predicates that are relations over integers and such that each vertex of their underlying infinite directed graph has a finite degree.
2. The predicate MSB_0 defined as follows. The predicate MSB_0 is true of x and y if the binary representation of y is obtained by zeroing the most significant bit of x. More formally
$$\mathrm{MSB}_0 = \{(x, x - 2^i) \mid x \in \mathbb{N}, \text{ and } i = \lfloor \log(x) \rfloor\} \ .$$

As an intermediate step toward a better comprehension of the Crane Beach conjecture for $\mathbf{FO}^2$, we propose to study the relationship between $<$ and $\mathcal{F}$, and present a Crane Beach result which is thus one predicate shy from showing the Crane Beach conjecture for $\mathbf{FO}^2$ over arbitrary numerical predicates.

The main result of this paper is a proof of the Crane Beach conjecture for each layer of the alternation hierarchy of the two-variable first-order logic equipped with the linear order and the finite-degree predicates.

Note that the general arbitrary numerical predicates in the statement would entail a long standing conjecture on the circuit complexity of the addition function. Thus, this result can be viewed as a uniform version of this circuit lower bound. This result immediately implies that this hierarchy is strict. This provides, to the best of our knowledge, the first example of a Crane Beach conjecture that applies to each level of an alternation hierarchy. Ramsey's Theorem for 3-hypergraphs will be our key combinatorics tool. This theorem indicates that the Crane Beach conjecture for $\mathbf{FO}^2$ hinges on the interaction between finite-degree predicates and the predicate MSB_0.

On the two-variable restriction. It is already known that the first-order logic with the "+" predicate satisfies the Crane Beach conjecture. Furthermore, the MSB_0 predicate is definable in first-order logic with the predicate "+" and the unary predicate $\{2^x \mid x \in \mathbb{N}\}$. The proof

of the Crane Beach conjecture for "+" predicate can be augmented to handle this extra unary numerical predicate. Therefore, we deduce that the first-order logic with the order and the MSB_0 predicate also satisfies the Crane Beach conjecture.

The case of finite-degree predicates is more intricate. Indeed, even if this class of predicates satisfies a form of locality, it is still not known if the Crane Beach conjecture hold for $\mathbf{FO}[<, \mathcal{F}]$. This class contains numerous expressive numerical predicates as the *translated bit predicate* which is true in positions (x, y) if the $(y-x)^{\text{th}}$ bit of x is a one. The Crane Beach conjecture may holds for finite-degree predicates but the classical proof, e.g. *collapse on active domain*, seems to fail [2, 18, 11].

Organization of the paper. Section 2 is dedicated to the necessary definitions. In Section 3 we present an Ehrenfeucht-Fraïssé game adapted to our context. We present in Section 4 our main result with immediate corollaries. The final section is dedicated to the proof.

2 Definitions

A finite word $u = u_0 \cdots u_{n-1}$ of A^* is represented by a relational structure on the set $\{0, \cdots, n-1\}$ over the vocabulary consisting of the *letter predicates* $\{\mathbf{a} \mid a \in A\}$ and of the *numerical predicates*. On the one hand, the letter predicate $\mathbf{a}$ is interpreted as the subset of all the positions labelled by the letter a. On the other hand, a numerical predicate interpretation only depends on the size n of the input word. Therefore, an interpretation of the predicate symbol $\mathbf{P}$ of arity k is a sequence $P = (P_n)_n$, where $P_n \subseteq \{0, \ldots, n-1\}^k$. Note that $\mathbf{P}$ is a syntactic object, while P is its interpretation. Furthermore a numerical predicate is said to be *uniform* if it can be seen as a relation on integers. More precisely, a numerical predicate $P = (P_n)_n$ of arity k is uniform if there exists an integer relation $Q \subseteq \mathbb{N}^k$ satisfying $Q \cap \{0, \ldots, n-1\}^k = P_n$. From now on, we do not distinguish numerical predicates from their interpretation and uniform predicates are seen as relations on integers. The class of all numerical predicates is denoted by Arb. Remark that the word *uniform* in this context is not related to the classical notion of *uniformity* in circuit complexity.

Examples

- The classical predicates $x < y$ or $x + y = z$ and $xy = z$ are numerical predicates and are uniforms.
- The predicate $x + y = \max$, where max is the last position of the word, is not uniform.

The logical formulae we consider are the first-order formulae over finite words. They are obtained with the following grammar:

$$\varphi = \mathbf{a}(x) \mid \mathbf{P}(x_1, \ldots, x_k) \mid \varphi \wedge \varphi \mid \neg\varphi \mid \exists x\ \varphi \ .$$

Here $x, x_1, x_2, x_3, \ldots$ denote first-order variables, which are interpreted by positions in the word. The letter predicate $\mathbf{a}(x)$, is interpreted by "the letter in position x is an a," and $\mathbf{P}(x_1, \ldots, x_k)$, is interpreted by "the predicate P is true on $(x_1, \ldots, x_k)$." As usual, the Boolean connectives $\wedge$ and $\neg$ are interpreted by "and" and "not," respectively, and $\exists x$ as a first-order existential quantification. We use the standard notation $u \models \varphi$ to signify that the word u satisfies the formula φ. We also denote by $u \models \varphi(i)$ if the formula $\varphi(x)$ is *true* when its free variable is interpreted by the integer $i < |u|$. The *quantifier depth* of a formula is the maximal number of nested quantifiers.

Let $\mathcal{P}$ be a class of numerical predicates. We denote by $\mathbf{FO}[\mathcal{P}]$ the class of first-order formulae that use numerical predicates in $\mathcal{P}$. We also denote by $\mathbf{FO}^2[\mathcal{P}]$ the subclass of

formulae of $\mathbf{FO}[\mathcal{P}]$ that use only two variables but allows the reuse of them. We say that a language L is definable in a fragment of logic if there exists a formula in this fragment such that L is the language of words satisfying this formula.

Example. The language $A^*aA^*bA^*cA^*$ can be described by the first-order formula

$$\exists x\ \exists y\ \exists z\ x < y < z \wedge \mathbf{a}(x) \wedge \mathbf{b}(y) \wedge \mathbf{c}(z)\ .$$

This formula uses three variables x, y and z. However, by reusing x we can rearrange it so that it uses two variables:

$$\exists x\ \mathbf{a}(x) \wedge \Big(\exists y\ x < y \wedge \mathbf{b}(y) \wedge \big(\exists x\ y < x \wedge \mathbf{c}(x)\big)\Big) \tag{1}$$

The alternation hierarchy of $\mathbf{FO}^2$ is also of interest here. To define formally the number of alternations of a formula, it is not possible to use prenex canonical normal form obtained by applying DeMorgan's laws to move negations past conjunctions, disjunctions and quantifiers. Indeed, these constructions increase the number of variables. That said, the number of alternations is still a relevant parameter that could be defined as follows: Consider the tree naturally associated to a formula, as the grammar previously exposed. For instance, formula (1) has "$\exists$" as a root and the atomic formulae as the leaf. In a two-variable first-order formula we count the maximal number of alternations between the root and the leaves once the negations have been pushed on to the leaves. A more precise definition could be found in the article [19]. We denote by $\mathbf{FO}^2_k[\mathcal{P}]$ the formulae of $\mathbf{FO}^2[\mathcal{P}]$ that have at most k quantifier alternations. The hierarchy induced by $\mathbf{FO}^2_k[<]$ is known to be strict [19] and its membership problems is decidable [12, 14]. Without loss of generality, we will always consider two-variable logic over predicates of arity at most 2.

3 Ehrenfeucht-Fraïssé game

One of the important tools for proving our main result is the Ehrenfeucht-Fraïssé game for two-variable logic. It is often used in the context of finite model theory to show certain inexpressibility results. Libkin's book [15] provides a good exposition. In this section, we present the Ehrenfeucht-Fraïssé game and briefly sketch a proof that the Crane Beach conjecture holds for $\mathbf{FO}^2_m[<, +1]$. This could be easily proved by using some algebraic descriptions of $\mathbf{FO}^2_m[<, +1]$ obtained by Kufleitner and Lauser [13] but we prove it using Ehrenfeucht-Fraïssé game as an introduction to our general result.

In the context of two-variable logic with a bounded number of alternations m and quantifier depth s, the associated Ehrenfeucht-Fraïssé game is defined as follows:

- The game is played by two players: *Spoiler* and *Duplicator*, on two relational structures. In our case, the relational structures are associated with the words u and v equipped with the letter predicates and a finite number of numerical predicates.
- The first round starts with Spoiler, who chooses either u or v and plays by putting a pebble on a position. Then Duplicator chooses the other word and puts a pebble on one of its positions.
- The subsequent rounds proceed as follows: each word is labelled by at most two pebbles. First, the two oldest pebbles are removed. Then, Spoiler plays on one structure and Duplicator on the other. If the relational structures induced by the two pairs of pebbles are not isomorphic, Spoiler wins.
- During all the game, Spoiler can change at most m times between the two words. Duplicator wins the game if he did not loose the game before the end of the s^{th} round.

We say that Spoiler has a *winning strategy* if he has a strategy that allows him to win the game whatever Duplicator plays. The following theorem is a well-known result that could be easily adapted, for instance, from the book [15].

▶ **Theorem 1.** *A language L belongs to $\mathbf{FO}^2_m[\mathcal{P}]$ if and only if there exist predicates $P^1, \dots, P^t \in \mathcal{P}$ and $s \in \mathbb{N}$ so that for any words $(u, v) \in L \times L^c$ Spoiler has a winning strategy for the two-pebble game with s rounds and m alternations on (u, v) over the predicates $P^1, \dots, P^t$.*

This theorem is our main interface to logic in order to establish Crane Beach-like results. The proof method we are going to sketch is a rather classical *back-and-forth* construction. As we mention before, the next result is also a direct consequence of known algebraic characterisations of these fragments [13].

▶ **Proposition 2.** *For any m, languages with a neutral letter in $\mathbf{FO}^2_m[<, +1]$ are definable in $\mathbf{FO}^2_m[<]$.*

Sketch of proof. Let L be a language definable in $\mathbf{FO}^2_m[<, +1]$ and assume that it has a neutral letter c. Thanks to Theorem 1, there exist integers s and $k \leqslant m$ such that Spoiler has a winning strategy for the two-pebble game with s rounds and k alternations on (u, v), with $(u, v) \in L \times L^c$. We construct two words u' and v' by inserting $2s$ letters c between each position (including the beginning and the end of the words). As c is a neutral letter, we have $(u', v') \in L \times L^c$ and therefore Spoiler has a winning strategy for the two-pebble game with s rounds and k alternations. Remark that the successor relation on (u', v') is useless since the non-neutral letters are not reachable from each other in less that s rounds. Therefore one can translate the Spoiler's wining strategy on (u', v') on a wining strategy that does not use the successor relation. This wining strategy can then be translated in a wining strategy on (u, v). We then conclude thank to Theorem 1. ◀

4 Main Result

We now investigate the Crane Beach conjecture in the specific case of $\mathbf{FO}^2$ equipped with numerical predicates of finite degree. Throughout this section, we borrow from the vocabulary of graph theory in order to express properties on the structure of numerical predicates. Indeed, a binary numerical predicate can be understood as a family of graphs. Furthermore, if the predicate is uniform, it can be viewed as a single infinite graph where the set of vertices is $\mathbb{N}$. Let P be a uniform numerical predicate. The *degree* of a position k for P, denoted by $d_P(k)$, is the size of the set of all integers connected to k *via* P. More formally

$$d_P(k) = |\{j \mid (k, j) \in P \text{ or } (j, k) \in P\}| \ .$$

The notion of locality is one of the most effective tools for using the Ehrenfeucht-Fraïssé games. One way of introducing locality is to restrict the degree of the signature. A uniform binary predicate P has a *finite degree* if all positions have a finite degree. We denote by $\mathcal{F}$ the class of binary uniform finite-degree predicates.

Examples

- The predicate $kx = y$, $x^k = y, \dots$ as well as the graph of any strictly growing function.
- The translated Bit predicate which is true in (x, y) if the $(y - x)^{\text{th}}$ bit of x is a one.

Example of nonfinite-degree predicates

- The linear ordering.
- The Bit predicate which is true of (x, y) if the y^{th} bit of x is a one.
- The MSB_0 predicate.

Predicates of finite degree do not include by definition uniform monadic predicates. However, all uniform monadic predicates can be encoded as predicates of finite degree. If P is monadic and uniform then $Q = \{(x, x) \mid x \in P\}$ is a finite-degree predicate.

The next theorem states that the Crane Beach conjecture for $\mathbf{FO}^2[\text{Arb}]$ reduces to solving the Crane Beach conjecture for the order, the MSB_0 predicate and the class of finite-degree predicates. The proof of this theorem is an adaptation of a circuit-version of a similar result [10]. Because of the lack of space, the proof of this theorem is omitted.

▶ **Theorem 3.** *Any language with neutral letter definable in* $\mathbf{FO}^2[\text{Arb}]$ *is definable in* $\mathbf{FO}^2[<, \mathcal{F}, \text{MSB}_0]$.

We believe that this last theorem does not hold without the neutral-letter hypothesis. For instance, the language $\{u\overline{u} \mid u \in A^*\}$, where $\overline{u}$ is the reversal image of u, is definable in $\mathbf{FO}^2[x+y = \max]$ but we conjecture that it is not definable by using only uniform predicates, and in particular, using predicates in the signature $[<, \mathcal{F}, \text{MSB}_0]$.

We now focus on the signature $[<, \mathcal{F}]$. To solve this problem, we will use the *locality* of the class $\mathcal{F}$. Locality is an effective tool which allows us to obtain numerous results of non-definability with the help of the Ehrenfeucht-Fraïssé games. Unfortunately, as soon as the order is present in the signature, it is no longer possible to use locality results and the absence of the order makes the fragment far less expressive. We are going to show that it is possible to add the order whilst conserving a form of locality when the other predicates are of finite degree.

▶ **Theorem 4** (Main Theorem). *Let* $m \geqslant 0$. *Any language with a neutral letter definable in* $\mathbf{FO}^2_m[<, \mathcal{F}]$ *is definable in* $\mathbf{FO}^2_m[<]$.

We immediately obtain the following corollary.

▶ **Corollary 5.** *Any language with a neutral letter definable in* $\mathbf{FO}^2[<, \mathcal{F}]$ *is definable in* $\mathbf{FO}^2[<]$.

This theorem states the uselessness of finite-degree predicates for defining languages with a neutral letter in two-variable logic. More precisely, they do not even improve the logical complexity of the languages. Therefore, we immediately deduce the strictness of this hierarchy. Indeed, we mainly use the known facts that $\mathbf{FO}^2_m[<]$ is a strict hierarchy (see [19]) and that each layer is stable by inverse image of morphisms. This latter fact is a requirement for having an equational description as given in the article [12]. Then, it is sufficient to take the inverse image of a language L that separates $\mathbf{FO}^2_{m+1}[<]$ from $\mathbf{FO}^2_m[<]$ by a morphism that maps a letter which is not in the alphabet of L to the empty word.

5 Proof of the main theorem

The principal ingredients are a notion of *locality*, the Ehrenfeucht-Fraïssé games and Ramsey's Theorem. For the remaining of the proof we fix $P^1, \ldots, P^t$ as predicates in $\mathcal{F}$. Our objective is to prove that for any language L with a neutral letter definable in $\mathbf{FO}^2_m[<, P^1, \ldots, P^t]$, there exists s such that for every words $u \in L$ and $v \notin L$, Spoiler has a winning strategy for the Ehrenfeucht-Fraïssé game with two pebbles, s rounds and m alternations on (u, v) and over the signature $\{<, +1\}$. The proof is decomposed as follows.

1. First, we introduce the notion of a *position's neighbourhood*.
2. Then, we define an equivalence relation between triples of disjoint neighbourhoods, which will allow us to define the different roles that these triples could play throughout the course of the game.
3. We then extract triples of so-called *well-typed* positions, with the help of Ramsey's Theorem for 3-hypergraphs.
4. Finally, we will inductively construct a winning strategy for Spoiler over the signature $\{<, +1\}$ that uses at most s rounds and m alternations. Proposition 2 allows us to conclude.

Let $E \subseteq \mathbb{N}^2$ be defined by $\{x, y\} \in E$ if, and only if, x and y are two positions connected by one of the predicates. More precisely, $\{x, y\} \in E$ if and only if

$$P^1(x, y) \vee P^1(y, x) \vee \cdots \vee P^t(x, y) \vee P^t(y, x) \ .$$

The graph $(\mathbb{N}, E)$ is the graph behind our reasoning. As each predicate is of finite degree, the graph $(\mathbb{N}, E)$ is also of finite degree. From this point on, we assume that the integer s (the number of rounds in the game) is fixed.

5.1 Definition of neighbourhood

For an integer i, the usual notion of r-neighbourhood is defined as the set of integers at distance r from i in $(\mathbb{N}, E)$. It captures the intuition that two integers with similar r-neighbourhoods cannot be distinguished in r applications of the predicates. Adding linear order to the predicates, any element between two given integers is *connected* by the order. Our specialized notion of neighbourhood thus distinguishes between the linear order and the other predicates; to this end, let us first introduce the *closure* of a finite set $F \subseteq \mathbb{N}$ as:

$$\mathrm{Cl}(F) = \{\min F, \min F + 1, \ldots, \max F\} \ .$$

Then, intuitively combining at each step the use of the predicates and that of the order, we define the 0-*neighbourhood* of $i \in \mathbb{N}$ as:

$$V(i, 0) = \mathrm{Cl}(\{i\} \cup \bigcup_{\substack{k' \leqslant i \leqslant k \\ \{k', k\} \in E}} \{k', k\}) \ .$$

and, inductively, the $(r + 1)$-*neighbourhood* of $i \in \mathbb{N}$ as:

$$V(i, r + 1) = \mathrm{Cl}(\bigcup_{j \in V(i,0)} V(j, r)) \ .$$

Less formally, the 0-neighbourhood of i is the set of positions j such that by moving a pebble inside this set it is possible to jump over i. We obtain immediatly that $V(i, r) \subseteq V(i, r + 1)$.

▶ **Lemma 6.** *For all integers i and k, $V(i, k)$ is finite.*

We now define the function $g_s \colon \mathbb{N} \to \mathbb{N}$ by $g_s(i) = \min V(i, s)$.

▶ **Lemma 7.** *We have $\lim_i g_s(i) = +\infty$.*

From this we immediately deduce the following corollary, which establishes the possibility of obtaining an arbitrarily large number of neighbourhoods that do not overlap.

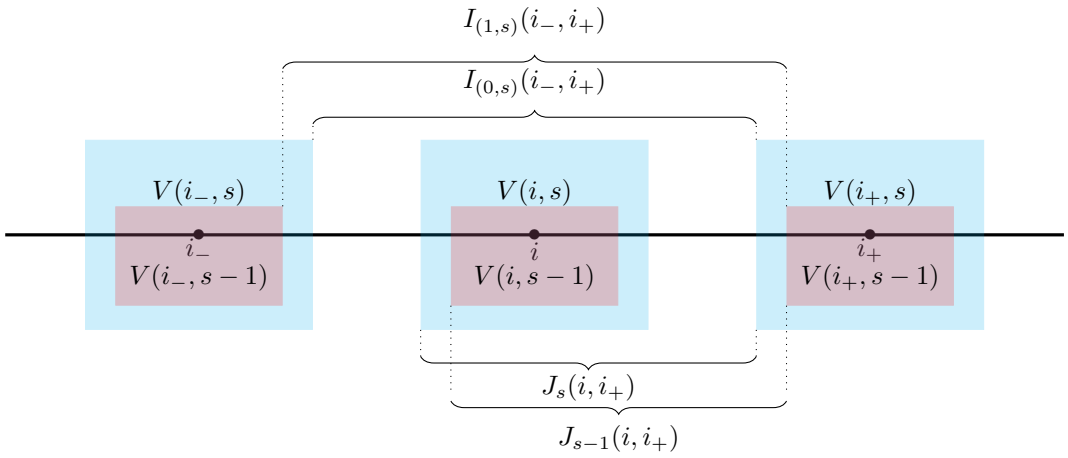


Figure 1 Neighbourhoods and segments.

▶ **Corollary 8.** *For any integer p, there exists $X \subseteq \mathbb{N}$ of size p such that for any $i, j \in X$, the s-neighbourhood of i and j are disjoint and separated by at least one integer.*

An *s-extraction* is a set of integers, such that their s-neighbourhoods are disjoint and separated by at least one integer. In short, they must be in accordance with the conditions of Corollary 8.

5.2 An equivalence relation for triples

We now introduce a notion of *similarity* for the triples of neighbourhoods taken from the Ehrenfeucht-Fraïssé two-pebble game. Let (i_-, i, i_+) be a triple of integers which is an s-extraction. More precisely, this triple satisfies that

1. $i_- < i < i_+$,
2. their s-neighbourhoods are disjoint and have at least one element between them.

According to Corollary 8, such a triple exists. We set $J_s(i, i_+)$ as the interval between the minimal position of the s-neighbourhood of i and minimal position of the s-neighbourhood of i_+. More formally,

$$J_s(i, i_+) = \{\min V(i, s), \ldots, \min V(i_+, s) - 1\} \ .$$

We also set $I_{(r,s)}(i_-, i_+)$ the interval in-between the maximal position of the $(s-r)$-neighbourhood of i_- and the minimal position of the $(s-r)$-neighbourhood of k. More formally

$$I_{(r,s)}(i_-, i_+) = \{\max V(i_-, s-r) + 1, \ldots, \min V(i_+, s-r) - 1\} \ .$$

These notations are illustrated in Figure 1.

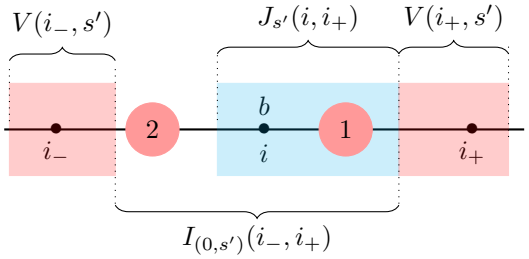
Let us take two triples (i_-, i, i_+) and (j_-, j, j_+) which form two s-extractions with $i_- < i < i_+$ and $j_- < j < j_+$. These two triples of integers are *equivalent* if two two-pebble *constrained games* are similar. We define two different notions of *constrained games* that differ only in their starting sets. These games only use two pebbles which are confined, at the r^{th} round, to the intervals

$$I_{(r,s)}(i_-, i_+) \text{ and } I_{(r,s)}(j_-, j_+) \ .$$

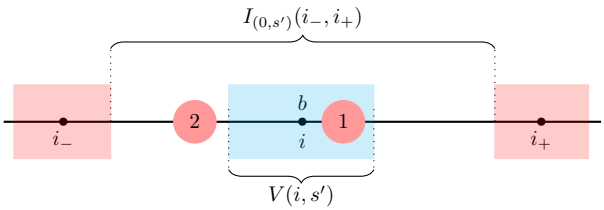
For the first game, the first pebble must be placed for both Spoiler and Duplicator in the sets $J_s(i, i_+)$ and $J_s(j, j_+)$. For the second game the first pebble is placed by Spoiler and Duplicator in the sets $V(i, s)$ and $V(j, s)$. If Duplicator wins these two games we can state that these two triples are equivalent, which we denote as $(i_-, i, i_+) \sim_s (j_-, j, j_+)$.

We now introduce formally this definition. We say that $(i_-, i, i_+) \sim_s (j_-, j, j_+)$ if for all $s' \leqslant s$ Duplicator wins the two following games. They are two-pebble games with s' rounds and s' alternation (we consider that Spoiler may alternate as much as he wishes between the two words) on the signature $\{<, P^1, \ldots, P^t\}$, and all positions are labelled by the same letter a with the exception of positions i and j which are labelled by the same letter b distinct from a. Here is the formal description of the two games:

1. For the first game, the first pebble of Spoiler and the first pebble of Duplicator are constrained to the set $J_{s'}(i, i_+)$ and $J_{s'}(j, j_+)$. At the r^{th} round, the players are constrained to choose positions in the sets $I_{(r,s')}(i_-, i_+)$ and $I_{(r,s')}(j_-, j_+)$.



2. For the second game, the Spoiler's first pebble and the first pebble of Duplicator are constrained to $V(i, s')$ and $V(j, s')$. At the r^{th} round, the players are constrained to play in the sets $I_{(r,s')}(i_-, i_+)$ and $I_{(r,s')}(j_-, j_+)$.



We say that positions $x \in I_{(r,s')}(i_-, i_+)$ and $y \in I_{(r,s')}(j_-, j_+)$ are *locally equivalent* if Duplicator can win the two restricted games when the pebbles are at these positions. The property presented in the following lemma can be deduced from the definitions and will be useful later.

▶ **Lemma 9.** *Let (i_-, i, i_+) an s-extraction. For every $0 \leqslant r \leqslant s$, we have the following*

$$J_{s-r}(i_-, i) \cup J_{s-r}(i, i_+) = V(i_-, s-r) \cup I_{(r,s)}(i_-, i_+) \ .$$

We now prove that $\sim_s$ is a finite-index equivalence relation. This is a rather classical result for this type of object in finite model theory. We remark that the equivalent classes can be seen as the sets of true formulae for each triple in a logic adapted to the two restricted games. Thus, two triples would be equivalent if they satisfy the same formulae of quantifier depth less than s. As the number of formulae is finite, we can easily deduce that $\sim_s$ equivalence relation.

▶ **Lemma 10.** *The relation $\sim_s$ is an equivalence relation of finite index.*

Ramsey's Theorem is a combinatorial result of graph theory often used in finite model theory. Here we use a version adapted to *hypergraphs*. We introduce it in the context of triples, which is a direct reformulation of the 3-*hypergraphs* variant. This theorem establishes that for every large hypergraph with coloured edges, it is possible to extract a sufficiently large monochrome sub-hypergraph. This theorem allows us to find an arbitrarily large set of triples which are all pairwise equivalent for the $\sim_s$ relation. For a set E, we denote by $\mathcal{P}_3(E)$ the set of pairwise disjoint triples of E.

▶ **Theorem 11** (Ramsey's Theorem for 3-hypergraphs [17]). *Let c be an integer. For any integer p there exists an integer n such that for any set S of size n and any function $h\colon \mathcal{P}_3(S) \to \{1, \dots, c\}$ there exists a set $F \subseteq S$ of size p such that h is constant on $\mathcal{P}_3(F)$.*

A well-typed s-extraction is a set X that is an s-extraction and such that all the triples of X are equivalent for $\sim_s$. The following corollary is an immediate from Ramsey's Theorem, in which c is the number of s-types of triples and h is the function that associates triple with their s-type.

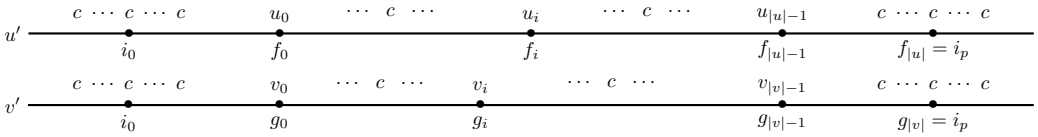
▶ **Corollary 12.** *For all integers p there exists a well-typed s-extraction of size p.*

We have now presented all of the tools necessary to present a proof of Theorem 4.

5.3 Core of the proof

Let L be a language with c as a neutral letter and definable in $\mathbf{FO}^2_m[<, P^1, \dots, P^t]$. According to Theorem 1, there exists an integer s, such that for any words $(u, v) \in L \times L^c$, Spoiler has a winning strategy for the two-pebble game with s rounds and m alternations for the signature $\{<, P^1, \dots, P^t\}$. Let (u, v) in $L \times L^c$ be such a pair. We now construct a strategy for Spoiler using only the order and the successor. Let $p = \max(|u|, |v|) + 1$. According to Corollary 12, there exists $X = \{i_0 < i_1 \cdots < i_p\}$, which is a well-typed s-extraction. Let $n = \max V(i_p, s)$, and let u' and v' be two words of length n and $(f_i)_{0 \leqslant i < |u|}, (g_i)_{0 \leqslant i < |v|}$ such that:

- $i_0 < f_0 < f_1 < \cdots < f_{|u|-1} < f_{|u|} = i_p$, and $i_0 < g_0 < g_1 < \cdots < g_{|v|-1} < g_{|v|} = i_p$,
- for all integers i, the positions f_i and g_i belong to X,
- $u'_{f_i} = u_i$, $v'_{g_i} = v_i$, $f_0 = g_0$ and $f_{|u|-1} = g_{|v|-1}$,
- all unassigned positions of u' and v' are labelled by the letter c.



If the words u and v are not of the same size, then that could give us $f_i \neq g_i$. The words u' and v' are nothing other than the words u and v after inserting neutral letters such that the non-neutral letters are on X. We also require the first and last non-neutral letters to be in the exact same positions.

As c is a neutral letter, (u', v') is in $L \times L^c$. Therefore, Spoiler has a winning strategy for the two-pebble game over s-round and m-alternation and the signature $\{<, P^1, \dots, P^t\}$. We now have to construct Spoiler's new strategy on (u, v). In order to do so, we simulate the game on (u', v') and construct *via* induction a winning strategy for Spoiler on (u, v). To achieve this step, we exploit a back-and-forth mechanism between the game on (u', v') and the game on (u, v). By following his winning strategy, Spoiler chooses a position on (u', v') which we translate into a position in (u, v). Duplicator then chooses a position in (u, v) which we translate on a position in (u', v'). We repeat this process until Duplicator can

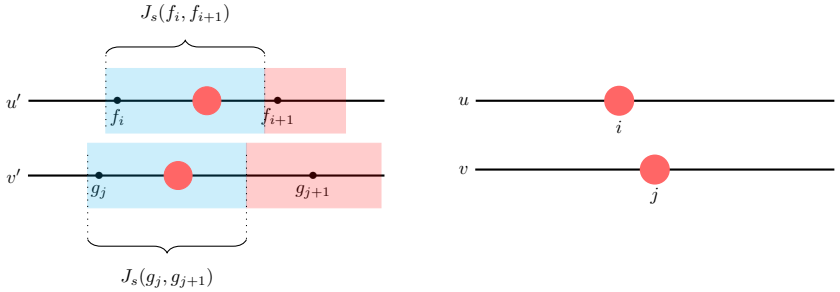
no longer respond in (u', v'). We must force Spoiler to play moves that are distant from one another so that his choices in (u', v') lead to a winning strategy on (u, v). If Spoiler's new pebble is in a neighbourhood different to that of the previous pebble, then by construction of the neighbourhoods, the numerical predicates, with the exception of the order predicate, do not allow for a connection between the two positions; they do not transmit *information*. In the following section we always denote by i_r (resp. j_r) the position of the pebble played at the round r on u (resp. v). Likewise, we use i'_r (resp. j'_r) for the position of the pebble at the round r on u' (resp. v').

For this construction to work, Spoiler should not win the game on (u', v') before he wins it on (u, v). This could however happen if Duplicator's choices on (u', v') are not pertinent. We avoid this situation by selecting locally equivalent positions, that is, positions where Duplicator wins the restricted games introduced in the preceding section. Thus, Spoiler cannot win by choosing moves that are close to the old pebbles. He is therefore forced to play some distant moves.

When Spoiler plays on an *extremal position* of the game on (u', v'), Duplicator can always respond at the same position on the other word. These moves therefore are of no interest in Spoiler's strategy. They are not used in the construction of the strategy of the game on (u, v). Each time Spoiler makes such a move, the game on (u, v) does not progress. More specifically, if the game has not started, the pebbles are not even placed and if the pebbles are already placed, they are not moved.

We begin by describing the game's first round, then we inductively build a strategy for the following rounds. For the first move, Spoiler's winning strategy designates a position for the game on (u', v'). Through symmetry, we assume that this is a position on u'. We therefore distinguish two cases:

1. This first move occurs within a segment of the form $J_s(f_i, f_{i+1})$ for an integer $0 \leqslant i < |u|$. In this case, we choose to play on the position i on the game on (u, v). Duplicator then responds in the game on (u, v) by playing on v at a position j. If the letter that marks j is different from the one that marks i, Duplicator loses the game immediately. Otherwise, we have to simulate Duplicator's response in the game on (u', v') by choosing a position in $J_s(g_j, g_{j+1})$ that is locally equivalent to Spoiler's first pebble. This is possible as the letters that mark f_i on u' and g_j on v' are equal, and $(f_{i-1}, f_i, f_{i+1}) \sim_s (g_{j-1}, g_j, g_{j+1})$.



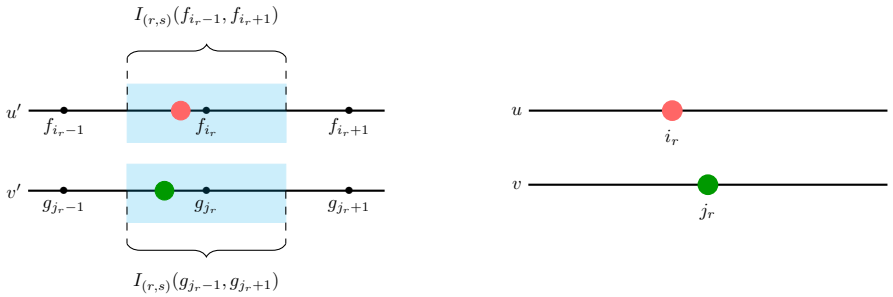
2. This first move is on an extremal position, that is smaller than $\min J_s(f_0, f_1) = \min J_s(g_0, g_1)$ or bigger than $\max J_s(f_{|u|-1}, f_{|u|}) = \max J_s(g_{|v|-1}, g_{|v|})$. In this case, the back-and-forth process is degenerate since the game on (u, v) has not started yet. It starts when Spoiler plays on a non-extremal position.
   This kind of moves is not useful for Spoiler since Duplicator can only answer on the game on (u', v') by choosing the exact same position on the other word. As long as Spoiler plays on these extremal positions, it is sufficient for Duplicator to choose the exact same position. As Spoiler follows a winning strategy, he eventually plays inside a segment $J_s(f_i, f_{i+1})$ for some integers $0 \leqslant i < |u|$. Indeed, the extremal positions together with

segments $J_s(f_i, f_{i+1})$ split into a partition of all positions of the word (see Figure 1). Therefore, we can assume to be in the preceding case.

We now explain how to construct a winning strategy for Spoiler on (u, v) for the next rounds. We construct it inductively. We now assume to have played $1 \leqslant r < s$ rounds and that the pebbles of the preceding round are on positions i_r on u (resp. j_r on v) as well as i'_r on u' (resp. j'_r on v'). It is Spoiler's turn to play. By induction, we assume the following properties to be satisfied:

- If positions i'_r and j'_r belong to $I_{(r,s)}(f_{i_r-1}, f_{i_r+1})$ and to $I_{(r,s)}(g_{j_r-1}, g_{j_r+1})$ then they are locally equivalent for at least one of the two constrained games at $(s-r)$-rounds (see Figure 2). The first constrained game corresponds to the second case, and the second constrained game corresponds to the third case.
- If this latter condition is not satisfied, then both pebbles have the exact same value, which is an extremal position on (u', v'). More precisely, $i'_r = j'_r$ and either

$$i'_r < \min J_{s-r}(f_0, f_1) = \min J_{s-r}(g_0, g_1) \text{ or}$$
$$i'_r > \max J_{s-r}(f_{|u|-1}, f_{|u|}) = \max J_{s-r}(g_{|v|-1}, g_{|v|}) \ .$$



- We assume the configuration of the game on (u', v') to be winnable for Spoiler: he has a winning strategy in less than $(s-r)$ rounds.

We are going to distinguish two cases. Either Duplicator is going to answer on Spoiler's latest move in the game on (u, v) or Spoiler wins the game. Since we seek a winning strategy for Spoiler, we assume that Duplicator successfully answers on (u, v). If this is true, then we are going to find an adequate answer for Duplicator in the game on (u', v'). Since Spoiler has a winning strategy for this latter game, Duplicator eventually loses the game on (u', v') and therefore the game on (u, v). We remark that the number of alternations of the new Spoiler's winning strategy on (u, v) is at most the one of his strategy on (u', v'). This concludes the proof.

Nevertheless, it remains to be explained how we construct the position of Spoiler on (u, v) and how to deduce from a correct answer for Duplicator on (u, v), a correct answer for Duplicator on (u', v').

We use the Spoiler's winning strategy on (u', v') to construct a new move for Spoiler on (u, v). Without loss of generality, we assume that this move is on u' and we denote by i'_{r+1} its position. We now distinguish four cases that only depend on the value of i'_{r+1} (see Figure 2). Indeed, the segment $\{0, \dots, n-1\}$ is split into four parts that correspond to the four following cases:

1. The first case corresponds to segments of the form $J_{s-r-1}(f_k, f_{k+1})$ for $k \neq i_r$ and $k \neq i_{r-1}$. It includes almost all the positions of $\{0, \dots, n\}$ except extremal positions and a *hole* around positions i'_r and i'_{r-1}.

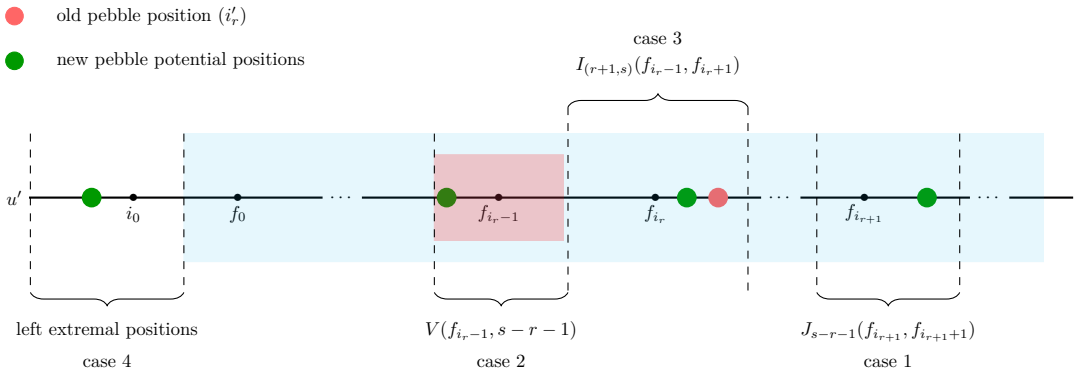


Figure 2 The four cases to deal with.

2. The second case corresponds to the *truncated* segment to the left of the previous pebble on u'. This is the initial segment of the second constrained game for this position. More precisely it is the segment $V(f_{i_r-1}, s-r-1)$.
3. The third case corresponds to the allowed positions for the constrained game around i'_r. More precisely, it is the segment $I_{(r+1,s)}(f_{i_r-1}, f_{i_r+1})$.
4. The last case corresponds to the extremal positions. They are the positions that are not handled by the other cases. They are either at the beginning or at the end of the word.

The four cases deal with all the positions since the segments of the form $J_{s-r-1}(f_k, f_{k+1})$ and the extremal positions form a partition of all the positions. Furthermore, by Lemma 9, we have

$$J_{s-r-1}(f_{i_r-1}, f_{i_r}) \cup J_{s-r-1}(f_{i_r}, f_{i_r+1}) = V(f_{i_r-1}, s-r-1) \cup I_{(r+1,s)}(f_{i_r-1}, f_{i_r+1}) \ .$$

We now construct the back-and-forth strategy for each of the four cases:

1. There exists an integer k different from i_r and $i_r - 1$ such that the position i'_{r+1} belongs to $J_{s-r-1}(f_k, f_{k+1})$. It is then sufficient for Spoiler to choose $i_{r+1} = k$ on u as its next move for the game on (u, v). We remark that all the predicates other than the linear order are evaluated to *false* between i'_r and i'_{r+1}. We assume Duplicator to be able to answer correctly at a position j_{r+1}. We now choose a position j'_{r+1} on v' in the set $J_{s-r-1}(g_{j_{r+1}}, g_{j_{r+1}} + 1)$ such that positions i'_{r+1} and j'_{r+1} are locally equivalent for the first constrained game. This is possible since positions $f_{i_{r+1}}$ and $g_{j_{r+1}}$ are labelled by the same letter and because

$$(f_{i_{r+1}-1}, f_{i_{r+1}}, f_{i_{r+1}+1}) \sim_s (g_{j_{r+1}-1}, g_{j_{r+1}}, g_{j_{r+1}+1}) \ .$$

   We remark that all predicates except for the linear order are evaluated as *false* between j'_r and j'_{r+1}. Furthermore, the value of the order predicate between i'_r and i'_{r+1} is exactly the same as between i_r and i_{r+1} which is also the same as between j_r and j_{r+1} and between j'_r and j'_{r+1}. Since the letters labelling positions i_{r+1} on u and j_{r+1} on v are the same, we deduce that position j'_{r+1} is correct for Duplicator. Consequently, the new configuration satisfies the induction hypothesis.
2. We assume that i'_{r+1} belongs to $V(f_{i_r-1}, s-r-1)$. In this case, we choose $i_{r+1} = i_r - 1$, meaning that Spoiler plays on the position just to the left of i_r. Since the successor relation is in the signature, Duplicator is also forced to play at the position immediately to the left. Here the very same arguments that in case 1 allow us to build a position j'_{r+1}

so that the new configuration satisfies the induction hypothesis hold. The only difference, is that this time we are using the second constrained game, not the first.

3. If i'_{r+1} belongs to $I_{(r+1,s)}(f_{i_r-1}, f_{i_r+1})$, then according to the induction hypothesis, Duplicator has a position j'_{r+1} in the set $I_{(r+1,s)}(g_{j_r-1}, g_{j_r+1})$ which is locally equivalent to i'_{r+1}. By choosing this position and by setting $i_{r+1} = i_r$ and $j_{r+1} = j_r$, we obtain a new configuration that satisfies the induction hypothesis. We remark that in this case, the game configuration on (u, v) does not change.
4. The last case is the one which i'_{r+1} does not satisfy any of the preceding case. By construction, the positions of the words are split into segments $J_{s-r}(f_k, f_{k+1})$ (resp. $J_{s-r}(g_k, g_{k+1})$) and the extremal positions. Therefore, if the integer i_{r+1} is not treated by the other cases, then this position has to be extremal. That is to say

$$i'_{r+1} < \min J_{s-r-1}(f_0, f_1) = \min J_{s-r-1}(g_0, g_1)$$

or

$$i'_{r+1} > \max J_{s-r-1}(f_{|u|-1}, f_{|u|}) = \max J_{s-r-1}(g_{|v|-1}, g_{|v|}) \ .$$

We choose $j'_{r+1} = i'_{r+1}$ for Duplicator on v', as well as $i_{r+1} = i_r$ and $j_{r+1} = j_r$. Therefore the game on (u, v) does not evolve and the new configuration satisfies the induction hypothesis. We remark that it is possible for i'_{r+1} to be an extremal position but be handled by one of the preceding cases. For instance, if i_r belongs to $J_{s-r}(f_0, f_1)$ and if

$$i_{r+1} \in I_{(r+1,s)}(i_0, f_1) \cap \{0, \dots, \min J_{s-r-1}(f_0, f_1)\} \ ,$$

then Duplicator follows the first constrained game and it is therefore possible that $i_{r+1} \neq j_{r+1}$. In this particular case, since i'_{r+1} and j'_{r+1} are locally equivalent, the configuration still satisfies the induction hypothesis.

As all the cases are treated, we have proved that as long as Duplicator answers correctly on (u, v), it is possible for him to answer correctly on (u', v'). Since Spoiler follows a winning strategy on (u', v'), Duplicator will eventually not be able to answer on (u, v). This concludes the proof.

Acknowledgement. I am grateful to Olivier Carton for all of his help and support, without which this paper would not have been possible. I also thank Michaël Cadilhac, Amy Hadfield, and the anonymous reviewers for their contribution in improving a lot the final version of this paper.

References

1 David A. Mix Barrington, Kevin Compton, Howard Straubing, and Denis Thérien. Regular languages in NC^1. *J. Comput. System Sci.*, 44(3):478–499, 1992.

2 David A. Mix Barrington, Neil Immerman, Clemens Lautemann, Nicole Schweikardt, and Denis Thérien. First-order expressibility of languages with neutral letters or: The Crane Beach conjecture. *J. Comput. System Sci.*, 70(2):101–127, 2005.

3 AshokK. Chandra, Steven Fortune, and Richard Lipton. Lower bounds for constant depth circuits for prefix problems. In Josep Diaz, editor, *Automata, Languages and Programming*, volume 154 of *Lecture Notes in Computer Science*, pages 109–117. Springer Berlin Heidelberg, 1983.

4 Shiva Chaudhuri and Jaikumar Radhakrishnan. Deterministic restrictions in circuit complexity. In *Proceedings of the Twenty-eighth Annual ACM Symposium on the Theory of Computing (Philadelphia, PA, 1996)*, pages 30–36. ACM, New York, 1996.

5 Merrick L. Furst, James B. Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17(1):13–27, 1984.

6 Neil Immerman. Languages that capture complexity classes. *SIAM J. Comput.*, 16(4):760–778, 1987.

7 Michal Koucký. Circuit complexity of regular languages. *Theory Comput. Syst.*, 45(4):865–879, 2009.

8 Michal Koucký, Clemens Lautemann, Sebastian Poloczek, and Denis Therien. Circuit lower bounds via ehrenfeucht-fraisse games. In *Proceedings of the 21st Annual IEEE Conference on Computational Complexity*, CCC'06, pages 190–201, Washington, DC, USA, 2006. IEEE Computer Society.

9 Michal Koucký, Pavel Pudlák, and Denis Thérien. Bounded-depth circuits: Separating wires from gates. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, STOC'05, pages 257–265, New York, NY, USA, 2005. ACM.

10 Andreas Krebs and Pierre Mckenzie. Private communication.

11 Andreas Krebs and A. V. Sreejith. Non-definability of languages by generalized first-order formulas over $(\mathbb{N},+)$. In *Proceedings of the 27th Annual IEEE/ACM Symposium on Logic in Computer Science*, LICS'12, pages 451–460, Washington, DC, USA, 2012. IEEE Computer Society.

12 Andreas Krebs and Howard Straubing. An effective characterization of the alternation hierarchy in two-variable logic. In *Proceedings of the 32th international conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'12)*, volume 18 of *LIPIcs – Leibniz International Proceedings in Informatics*, pages 86–98, Dagstuhl, Germany, 2012. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

13 Manfred Kufleitner and Alexander Lauser. Quantifier alternation in two-variable first-order logic with successor is decidable. In *Proceedings of the 30th International Symposium on Theoretical Aspects of Computer Science*, STACS'13, pages 305–316. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2013.

14 Manfred Kufleitner and Pascal Weil. The FO^2 alternation hierarchy is decidable. In *Computer science logic 2012*, volume 16 of *LIPIcs – Leibniz International Proceedings in Informatics*, pages 426–439. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012.

15 Leonid Libkin. *Elements of Finite Model Theory*. Springer, 2004.

16 Prabhakar Ragde and Avi Wigderson. Linear-size constant-depth polylog-threshold circuits. *Inform. Process. Lett.*, 39(3):143–146, 1991.

17 Frank P. Ramsey. On a Problem of Formal Logic. *Proc. London Math. Soc.*, S2-30(1):264, 1930.

18 Amitabha Roy and Howard Straubing. Definability of languages by generalized first-order formulas over $(\mathbb{N},+)$. *SIAM J. Comput.*, 37(2):502–521 (electronic), 2007.

19 Philipp Weis and Neil Immerman. Structure theorem and strict alternation hierarchy for FO^2 on words. *Log. Methods Comput. Sci.*, 5(3):3:4, 23, 2009.

Two-variable Logic with Counting and a Linear Order

Witold Charatonik and Piotr Witkowski

Institute of Computer Science
University of Wroclaw, Poland
{wch,pwit}@cs.uni.wroc.pl

Abstract

We study the finite satisfiability problem for the two-variable fragment of the first-order logic extended with counting quantifiers (C^2) and interpreted over linearly ordered structures. We show that the problem is undecidable in the case of two linear orders (in presence of two other binary symbols). In the case of one linear order it is NExpTime-complete, even in presence of the successor relation. Surprisingly, the complexity of the problem explodes when we add one binary symbol more: C^2 with one linear order and its successor, in presence of other binary predicate symbols, is decidable, but it is as expressive (and as complex) as Vector Addition Systems.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases Two-variable logic, counting quantifiers, linear order, satisfiability, complexity

Digital Object Identifier 10.4230/LIPIcs.CSL.2015.631

1 Introduction

Since 1930s, when Alonzo Church and Alan Turing proved that the satisfiability problem for first-order logic is undecidable, much effort was put to find decidable subclasses of this logic. One of the most prominent decidable cases is the two-variable fragment FO^2. FO^2 is particularly important in computer science because of its decidability and connections with other formalisms like modal, temporal or description logics or applications in XML or ontology reasoning. The satisfiability of FO^2 was proved to be decidable in [31, 23] and NExpTime-complete in [8].

All decidable fragments of first-order logic have a limited expressive power and a lot of effort is being put to extend them beyond the first-order logic while preserving decidability. Many extensions of FO^2, in particular with transitive closure or least fixed-point operators, quickly lead to undecidability [7, 11]. Extensions that go beyond the first order logic and enjoy decidable finite satisfiability problem include FO^2 over restricted classes of structures where one [15] or two relation symbols [16] are interpreted as equivalence relations; where one [25] or two relations are interpreted as linear orders [30]; where two relations are interpreted as successors of two linear orders [20, 6, 4]; where one relation is interpreted as linear order and another one as equivalence [1]; where one relation is transitive [33]; where an equivalence closure can be applied to two binary predicates [14]; where deterministic transitive closure can be applied to one binary relation [3]. It is known that the finite satisfiability problem is undecidable for FO^2 with two transitive relations [13], with three equivalence relations [15], with one transitive and one equivalence relation [16], with three linear orders [12], with two linear orders and their two corresponding successors [20]. A summary of complexity results for extensions of FO^2 with order relations can be found in [21].

24th EACSL Annual Conference on Computer Science Logic (CSL 2015).
Editor: Stephan Kreutzer; pp. 631–647

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The two-variable fragment with counting quantifiers (C^2) extends FO^2 by allowing counting quantifiers of the form $\exists^{<k}$, $\exists^{\leq k}$, $\exists^{=k}$, $\exists^{\geq k}$ and $\exists^{>k}$, for all natural numbers k. The two problems of satisfiability and finite satisfiability for C^2 (which are two different problems as C^2 does not have a finite model property) were proved to be decidable in [10]. Another solution of the satisfiability problem together with NExpTime-completeness result under unary encoding of numbers in counting quantifiers can be found in [26]. Pratt-Hartmann in [27] established NExpTime-completeness of both satisfiability and finite satisfiability under binary encoding of numbers in counting quantifiers. All these algorithms are quite sophisticated, a significant simplification can be found in [28]. There are not many known decidable extensions of C^2. In [4] it is shown that finite satisfiability for C^2 interpreted over structures where two binary relations are interpreted as forests of finite trees (which subsumes the case of two successor relations on two linear orders) is NExpTime-complete. [29] shows that the satisfiability and finite satisfiability problems for C^2 with one equivalence relation are both NExpTime-complete.

In this paper we study the extensions of C^2 with linear orders. We show that the finite satisfiability problem for C^2 with two linear orders, in presence other binary predicate symbols, is undecidable. For C^2 with one linear order, even if the successor of this linear order is present, in absence of other binary predicate symbols, it is decidable and NExpTime-complete. A surprising result is that when we add one more binary predicate symbol, the complexity of the problem explodes: C^2 with one linear order and its successor, in presence of other binary predicate symbols, is as expressive (and as complex) as multicounter automata.

Multicounter automata (MCA) is a very simple formalism equivalent to Petri Nets and vector addition systems (VAS) [24], which are used e. g., to describe distributed, concurrent systems and chemical/biological processes. One of the main reasoning tasks for VAS is to determine reachability of a given vector. It is known that this problem is decidable [17, 22, 18] and ExpSpace-hard [19], but precise complexity is not known, and after over 40 years of research it is even not known if the problem is elementary. We give a reduction from the emptiness problem of MCA (which is equivalent to the reachability for VAS) to finite satisfiability of C^2 with one linear order and its successor, in presence of one more binary predicate symbol. Although we show that C^2 with one linear order and its successor, in presence of arbitrary number of binary predicate symbols is decidable, it is very unlikely that it has an elementary decision algorithm since existence of such an algorithm implies existence of an elementary algorithm for VAS reachability.

Due to space limits we have omitted most proofs. The missing proofs can be found in the full version of the paper.

2 Preliminaries

We will consider finite satisfiability problems for the two-variable logic with counting (C^2 for short) over finite structures, where some binary symbols are interpreted as linear orders or successors of linear orders. W.l.o.g. we will be interested in largest antireflexive relations $<$ contained in linear orders $\leq$; for a linear order $\leq$ we write $a < b$ iff $a \leq b$ and $a \neq b$. We overload notation and name these relations $<$ linear orders. We use symbols $<$, $<_1$, $<_2$ to denote linear orders and $\dashv$, $\dashv_1$, $\dashv_2$ to denote their resp. successors. Given a finite vocabulary Σ we write $\mathcal{O}(\Sigma, <, \dashv)$ to denote the class of finite structures on vocabulary Σ, where $<$ and $\dashv$ have appropriate interpretation, and we adopt a similar notation for other classes of structures. Logics we consider will be denoted by $C^2[U, B, I]$, where U and B are vocabularies of resp. unary and binary symbols allowed in formulas, and $I \subseteq B$ is

the vocabulary of interpreted binary symbols. When $B = I$ we write $\mathrm{C}^2[U, I]$ instead of $\mathrm{C}^2[U, I, I]$.

Let Σ_u and Σ_b be countably infinite vocabularies of resp. unary and binary symbols and such that $\{<, +1, <_1, +1_1, <_2, +1_2, =\} \subseteq \Sigma_b$. Specifically, we will be interested in the following logics: $\mathrm{C}^2[\Sigma_u, \Sigma_b, \{<\}]$, $\mathrm{C}^2[\Sigma_u, \Sigma_b, \{<, +1\}]$ and $\mathrm{C}^2[\Sigma_u, \{<_1, s_1, <_2, s_2\}, \{<_1, <_2\}]$, where s_1 and s_2 are some binary symbols. By $\mathrm{C}^2[\Sigma_u, \Sigma_b, \{<\}]$ we mean the logic C^2 where vocabulary of every formula is a finite subset of $\Sigma_u \cup \Sigma_b$ and $<$ is interpreted as a linear order. Definition of $\mathrm{C}^2[\Sigma_u, \Sigma_b, \{<, +1\}]$ is similar, with the exception that $+1$ is interpreted as successor of $<$. The logic $\mathrm{C}^2[\Sigma_u, \{<_1, s_1, <_2, s_2\}, \{<_1, <_2\}]$ allows arbitrary number of unary and at most 4 binary symbols, two of them are interpreted as linear orders. Notice that we do not allow constant symbols in vocabularies, but this does not cause loss of generality since constants can be simulated by unary predicates and counting quantifiers.

3 Two linear orders

We start with an observation that the successor relation of a linear order can be expressed in $\mathrm{C}^2[\Sigma_u, \Sigma_b, \{<\}]$. More precisely, let s be a free binary symbol. The following lemma says that s can be defined to mean the successor of $<$ in $\mathrm{C}^2[\Sigma_u, \{<, s\}, \{<\}]$. Intuitively, it is enough to express that s is a subrelation of $<$ such that each node (with the exception of the least and the greatest one) has exactly one s-successor and exactly one s-predecessor.

▶ **Lemma 1.** *There exists a formula φ_s of $\mathrm{C}^2[\Sigma_u, \{<, s\}, \{<\}]$ such that for every finite structure $\mathcal{M}$, we have $\mathcal{M} \models \varphi_s$ if and only if $s^{\mathcal{M}}$ is the successor relation of $<^{\mathcal{M}}$.*

▶ **Corollary 2.** *Finite satisfiability of $\mathrm{C}^2[\Sigma_u, \Sigma_b, \{<, +1\}]$ is reducible in constant time to finite satisfiability of $\mathrm{C}^2[\Sigma_u, \Sigma_b, \{<\}]$.*

▶ **Corollary 3.** *Finite satisfiability of $\mathrm{C}^2[\Sigma_u, \{<_1, +1_1, <_2, +1_2\}, \{<_1, +1_1, <_2, +1_2\}]$ is reducible in constant time to finite satisfiability of $\mathrm{C}^2[\Sigma_u, \{<_1, s_1, <_2, s_2\}, \{<_1, <_2\}]$, where s_1 and s_2 are some binary symbols.*

Since $\mathrm{FO}^2[\Sigma_u, \{<_1, +1_1, <_2, +1_2\}, \{<_1, +1_1, <_2, +1_2\}]$, i. e., the two-variable logic with two linear orders and their corresponding successors, is undecidable [20], we have the following conclusion.

▶ **Corollary 4.** *Finite satisfiability problem of $\mathrm{C}^2[\Sigma_u, \Sigma_b, \{<_1, <_2\}]$ is undecidable. This remains true even for $\mathrm{C}^2[\Sigma_u, \{<_1, s_1, <_2, s_2\}, \{<_1, <_2\}]$, where s_1 and s_2 are distinct binary symbols.*

4 $\mathrm{C}^2[\Sigma_u, \{<, +1\}, \{<, +1\}]$ is NExpTime-complete

We will show that finite satisfiability problem for $\mathrm{C}^2[\Sigma_u, \{<, +1\}, \{<, +1\}]$ is NExpTime-complete. Since the lower bound follows from the complexity of FO^2 with only unary predicates [5, Theorem 11], we will concentrate on proving the upper bound. The proof presented here is similar to a corresponding result [2] on FO^2 on finite trees.

We assume that the input $\mathrm{C}^2[\Sigma_u, \{<, +1\}, \{<, +1\}]$ formula φ is in a normal form

$$\varphi = \forall x \forall y.\chi(x, y) \wedge \bigwedge_{h=1}^{m} \forall x \exists^{\preccurlyeq_h C_h} y.\chi_h(x, y),$$

where $\chi, \chi_1, \ldots, \chi_m$ are quantifier-free formulas with arbitrary unary predicates and binary predicates $<, +1$, symbols $\preccurlyeq_h \in \{\leq, \geq\}$ for $h = \{1, \ldots, m\}$, and $C_1, \ldots, C_m$ are positive

integers encoded in binary. For the rest of this section the constant c is fixed and it equals $\max\{C_h \mid h \in \{1,\ldots,m\}\}$. It is well known [9, Theorem 2.2] that by adding additional unary predicates each C^2 formula φ can be transformed in polynomial time to an equisatisfiable formula in normal form.

Observe that $\mathrm{C}^2[\Sigma_u, \{<, +1\}, \{<, +1\}]$ may be seen as a fragment of the weak monadic second-order logic with one successor WS1S, where unary relations are simulated by second-order existential quantifiers and counting quantifiers by first-order ones (e.g., a formula of the form $\exists^{\leq k} x.\chi(x)$ can be replaced by an equivalent formula with $k+1$ universal quantifiers). However, this view leads to formulas with three alternations of quantifiers that can be checked for satisfiability in 4ExpTime, which is not a desired complexity bound.

Because an element of a model of φ may require up to c witnesses for satisfaction, we will be interested in multisets counting these witnesses. Let $\mathbb{N}_c = \{n \in \mathbb{N} \mid n \leq c\} \cup \{\infty\}$. For $k, k' \in \mathbb{N}_c$ define $\mathrm{cut}_c(k) = k$ if $k \leq c$ and $\mathrm{cut}_c(k) = \infty$ if $k > c$. Define $k \oplus_c k' = \mathrm{cut}_c(k + k')$. A c-multiset of elements from a given set A is any function $f : A \to \mathbb{N}_c$. For a given element a in A, by $\{a\}$ we denote the multiset defined by $\{a\}(x) = 1$ if $x = a$ and $\{a\}(x) = 0$ for $x \neq a$. The union of two multisets f and g is a function denoted $f \cup g$ such that $(f \cup g)(x) = f(x) \oplus_c g(x)$. Empty multiset denoted $\emptyset$ is the constant function equal 0 for all arguments.

Let us call maximal consistent formulas specifying the relative position of a pair of nodes in a structure in $\mathcal{O}(\Sigma, <, +1)$ *order formulas*. There are five possible order formulas: $x{=}y \wedge \neg +1(y,x) \wedge \neg +1(x,y) \wedge y{\not<}x \wedge x{\not<}y$, $x{\neq}y \wedge +1(y,x) \wedge \neg +1(x,y) \wedge y{<}x \wedge x{\not<}y$, $x{\neq}y \wedge +1(x,y) \wedge \neg +1(y,x) \wedge x{<}y \wedge y{\not<}x$, $x{\neq}y \wedge \neg +1(x,y) \wedge \neg +1(y,x) \wedge x{<}y \wedge y{\not<}x$, and $x{\neq}y \wedge \neg +1(x,y) \wedge \neg +1(y,x) \wedge y{<}x \wedge x{\not<}y$. They are denoted, respectively, as: $\theta_=$, θ_{-1}, θ_{+1}, $\theta_<$, $\theta_>$. Let Θ be the set of these five formulas.

A *1-type* over the signature Σ is a maximal consistent conjunction of atomic and negated atomic formulas over Σ involving only the variable x. The set of all 1-types over Σ will be denoted $\Pi(\Sigma)$. The family of all multisets of 1-types over the signature Σ is denoted $\mathbb{N}_c^{\Pi(\Sigma)}$.

▶ **Definition 5** (Full type over Σ w.r.t. c)**.** A *full type* over Σ w.r.t. c is a function $\sigma : \Theta \to \mathbb{N}_c^{\Pi(\Sigma)}$, such that $\sigma(\theta_{-1})$ and $\sigma(\theta_{+1})$ are singletons or empty, and $\sigma(\theta_=)$ is a singleton.

▶ **Definition 6** (Full type in $\mathcal{A}$ w.r.t. c)**.** Let $\mathcal{A}$ be a structure over a vocabulary Σ and let a be an element of $\mathcal{A}$. A *full type* of a in $\mathcal{A}$, denoted $\mathrm{ft}^{\mathcal{A}}(a)$ is a function $\sigma : \Theta \to \mathbb{N}_c^{\Pi(\Sigma)}$ such that

- $\sigma(\theta_=)$ is the singleton of the 1-type of a in $\mathcal{A}$,
- $\sigma(\theta_{-1})$ is the singleton of the 1-type of the predecessor of a (if a has a predecessor) or empty multiset (if a has no predecessor),
- $\sigma(\theta_{+1})$ is the singleton of the 1-type of the successor of a (if a has a successor) or empty multiset (if a has no successor),
- $\sigma(\theta_<)$ is the c-multiset of 1-types of elements strictly smaller than a in $\mathcal{A}$, excluding the predecessor (if it exists), and
- $\sigma(\theta_>)$ is the c-multiset of 1-types of elements strictly greater than a in $\mathcal{A}$, excluding the successor (if it exists).

A structure $\mathcal{A}$ is said to realise a full type σ if $\mathrm{ft}^{\mathcal{A}}(a) = \sigma$ for some $a \in \mathcal{A}$.

In the following, we often identify a full type σ, which is a function, with the tuple $\langle \sigma(\theta_{-1}), \sigma(\theta_=), \sigma(\theta_{+1}), \sigma(\theta_<), \sigma(\theta_>)\rangle$. We define Σ-c-graph as the graph $\langle V, E\rangle$ where the set V of nodes is the set of full-types over Σ w.r.t. c and the set E of edges is defined as follows.

$$\langle\langle \Pi_{-1}, \{\pi\}, \{\pi_{+1}\}, \Pi_<, \Pi_>\rangle, \langle \{\pi\}, \{\pi_{+1}\}, \Pi'_{+1}, \Pi'_<, \Pi'_>\rangle\rangle \in E \quad \text{iff} \quad \Pi'_< = \Pi_< \cup \Pi_{-1} \text{ and } \Pi_> = \Pi'_> \cup \Pi'_{+1}$$

Let σ be a full type such that $\sigma(\theta_=) = \{\pi\}$ and let $\forall x \exists^{\leqslant_h C_h} y.\chi_h(x,y)$ be a conjunct in φ. The following five functions are used to count witnesses w.r.t. this conjunct for elements of full type σ.

$$W_{=}^{\chi_h}(\sigma) = \begin{cases} 1 & \text{if } \pi(x) \models \chi_h(x,x) \\ 0 & \text{otherwise} \end{cases}$$

$$W_{-1}^{\chi_h}(\sigma) = \begin{cases} 1 & \text{if } \sigma(\theta_{-1}) = \{\pi'\} \text{ and } \pi(x) \wedge \pi'(y) \wedge \theta_{-1}(x,y) \models \chi_h(x,y) \\ 0 & \text{otherwise} \end{cases}$$

$$W_{+1}^{\chi_h}(\sigma) = \begin{cases} 1 & \text{if } \sigma(\theta_{+1}) = \{\pi'\} \text{ and } \pi(x) \wedge \pi'(y) \wedge \theta_{+1}(x,y) \models \chi_h(x,y) \\ 0 & \text{otherwise} \end{cases}$$

$$W_{<}^{\chi_h}(\sigma) = \mathrm{cut}_c\Big(\sum_{\pi':\pi(x)\wedge\pi'(y)\wedge\theta_<(x,y)\models\chi_h(x,y)} (\sigma(\theta_<))(\pi') \Big)$$

$$W_{>}^{\chi_h}(\sigma) = \mathrm{cut}_c\Big(\sum_{\pi':\pi(x)\wedge\pi'(y)\wedge\theta_>(x,y)\models\chi_h(x,y)} (\sigma(\theta_>))(\pi') \Big)$$

Note that in the definition above $(\sigma(\theta_>))(\pi')$ is simply the number of occurrences of the 1-type π' in the multiset $\sigma(\theta_>)$.

▶ **Definition 7** (Compatible full types)**.** We say that a full type σ such that $\sigma(\theta_=) = \{\pi\}$ is compatible with formula φ if the following conditions are satisfied.

- $\pi(x) \models \chi(x,x)$,
- $\pi(x) \wedge \pi'(y) \wedge \theta(x,y) \models \chi$ for all $\theta \in \{\theta_{-1}, \theta_{+1}, \theta_<, \theta_>\}$ and all $\pi' \in \sigma(\theta)$, and
- for each conjunct $\forall x \exists^{\leqslant_h C_h} y.\chi_h(x,y)$ of φ we have

$$W_{=}^{\chi_h}(\sigma) + W_{+1}^{\chi_h}(\sigma) + W_{-1}^{\chi_h}(\sigma) + W_{>}^{\chi_h}(\sigma) + W_{<}^{\chi_h}(\sigma) \leqslant_h C_h$$

It is quite obvious that whenever $\mathcal{A} \models \varphi$, all full types realised in $\mathcal{A}$ are compatible with φ. It is not difficult to see that the converse is also true, as the following lemma says.

▶ **Lemma 8.** *For any ordered structure $\mathcal{A}$ and any $\mathrm{C}^2[\Sigma_u, \{<, +1\}, \{<, +1\}]$ formula φ in normal form, if all full types realised in $\mathcal{A}$ are compatible with φ then $\mathcal{A} \models \varphi$.*

We define Σ-c-φ-graph as the subgraph of Σ-c-graph consisting of nodes compatible with φ. The nodes of the form $\langle \emptyset, \ldots, \ldots, \emptyset, \ldots \rangle$ are called *source* nodes; the nodes of the form $\langle \ldots, \ldots, \emptyset, \ldots, \emptyset \rangle$ are called *target* nodes. Intuitively, a source node corresponds to a full type of the least element in some model of φ while a target node corresponds to the greatest element in some model.

▶ **Lemma 9.** *Let φ be a $\mathrm{C}^2[\Sigma_u, \{<, +1\}, \{<, +1\}]$ formula in normal form over vocabulary Σ. Formula φ is finitely satisfiable if and only if there exists a path from a source node to a target node in Σ-c-φ-graph.*

Lemma 9 leads us directly to the main theorem of this section. To check satisfiability of a formula in $\mathrm{C}^2[\Sigma_u, \{<, +1\}, \{<, +1\}]$ it is enough to guess an appropriate path in Σ-c-φ-graph. Moreover, it is enough to use only exponentially many different full types in the guessed path.

▶ **Theorem 10.** *The finite satisfiability problem for $\mathrm{C}^2[\Sigma_u, \{<, +1\}, \{<, +1\}]$ is* NExpTime*-complete.*

Proof. The lower bound follows from the complexity of FO^2 with only unary predicates. For the upper bound, an algorithm for deciding finite satisfiability of $\mathrm{C}^2[\Sigma_u, \{<, +1\}, \{<, +1\}]$ works as follows. We take a $\mathrm{C}^2[\Sigma_u, \{<, +1\}, \{<, +1\}]$ formula φ and convert it to an equisatisfiable normal form (in polynomial time) if necessary. Then we guess a path from a source node to a target node in Σ-c-φ-graph where Σ is the vocabulary of φ. This requires in particular verification of the fact that all nodes are compatible with φ. All this can be accomplished in time polynomial in the size of the graph. This size is potentially doubly exponential in $|\varphi|$: the number of all 1-types over Σ is exponential in $|\varphi|$, so the number of sets of 1-types, and, in consequence, the number of full types, is doubly exponential. The potential 2NExpTime complexity of the algorithm can be lowered to NExpTime using the observation that the $\theta_<$ and $\theta_>$ components of full types behave in a monotone way along any path connecting any source node with any target node. The $\theta_<$ component may only increase and $\theta_>$ only decrease along any such path. Since a multiset may increase only exponentially many times (and only exponentially many times it may decrease) there are only exponentially many such multisets occurring along the path. Therefore it is enough to guess only exponentially many different full types. ◀

5 Hardness of $\mathrm{C}^2[\Sigma_u, \{<, +1, s\}, \{<, +1\}]$

We will show that finite satisfiability problem for $\mathrm{C}^2[\Sigma_u, \{<, +1, s\}, \{<, +1\}]$, where s is a binary relation, is at least as hard as non-emptiness of multicounter automata. Below, for a given MCA M we construct a $\mathrm{C}^2[\Sigma_u, \{<, +1, s\}, \{<, +1\}]$ formula φ_M which has a finite model if and only if M is non-empty.

Multicounter automata

We adopt a notion of *multicounter automata* (MCA for short) similar to one in [32], but with empty input alphabet and simplified counter manipulation. Intuitively, a MCA is a finite state automaton without input but equipped with a finite set of counters which can be incremented and decremented, but not tested for zero. More formally, a multicounter automaton M is a tuple $\langle Q, C, R, \delta, q_I, F\rangle$, where the set Q of states, the initial state $q_I \in Q$ and the set $F \subseteq Q$ of final states are as in usual finite state automata, C is a finite set (the *counters*) and R is a subset of C. The transition relation δ is a subset of

$$Q \times \{inc(c), dec(c), skip \mid c \in C\} \times Q.$$

An MCA is called *reduced* if it does not have skip transitions and $R = C$ (in this case we just omit R component of tuple M).

A *configuration* of a multicounter automaton M is a pair $\langle p, \vec{n}\rangle$ where p is a state and $\vec{n} \in \mathbb{N}^C$ gives a value $\vec{n}(c)$ for each counter c in C. Transitions with $inc(c)$ and *skip* can always be applied, whereas transitions with $dec(c)$ can only be applied to configurations with $\vec{n}(c) > 0$. Applying a transition $\langle p, inc(c), q\rangle$ to a configuration $\langle p, \vec{n}\rangle$ yields a configuration $\langle q, \vec{n}_0\rangle$ where $\vec{n}_0$ is obtained from $\vec{n}$ by incrementing its c-th component and keeping values of all other components unchanged. Analogously, applying (an applicable) transition $\langle p, dec(c), q\rangle$ to a configuration $\langle p, \vec{n}\rangle$ yields a configuration $\langle q, \vec{n}_0\rangle$ where $\vec{n}_0$ is obtained from $\vec{n}$ by decrementing its c-th component. Transitions with *skip* do not change value of any counter in C. A *run* is an interleaving sequence of configurations and transitions $conf_1, trans_1, \ldots, trans_{k-1}, conf_k$ such that $trans_i$ applied to $conf_i$ gives $conf_{i+1}$, for $1 \leq i < k$. A run is *accepting*, if it starts in configuration $\langle q_I, \vec{0}\rangle$ and ends in some configuration $\langle q_F, \vec{n}_F\rangle$ with $q_F \in F$ and $\vec{n}_F(c) = 0$ for every $c \in R$. The emptiness problem for multicounter automata is the question whether

a given automaton M has an accepting run. It is well known that this problem (for both MCA and reduced MCA) is decidable, as it is polynomial-time equivalent to reachability problem in Vector Addition Systems/Petri Nets[17, 22].

▶ **Definition 11.** Let $M = \langle Q, C, \delta, q_I, F\rangle$ be a reduced MCA. Let $\Sigma = \{q \mid q \in Q\} \cup \{inc_c, dec_c \mid c \in C\} \cup \{\min, \max, <, +1, s\}$ where predicates $q, inc_c, dec_c, \min$ and max are unary and $<, +1$ and s are binary. Define φ_M as the conjunction of the following Σ-formulas.

1. $\exists^{=1}x.\, \min(x) \wedge \exists^{=1}x.\, \max(x)$
2. $\forall x \forall y.\, (\min(x) \to (x < y \vee x = y)) \wedge (\max(x) \to (y < x \vee y = x))$
3. $\forall x.\, \Big(\bigvee_{q\in Q} q(x)\Big) \wedge \bigwedge_{q\in Q} \Big(q(x) \to \bigwedge_{q'\in Q\setminus\{q\}} \neg q'(x)\Big)$
4. $\forall x.\, (\min(x) \to q_I(x)) \wedge \Big(\max(x) \to \bigvee_{q_F\in F} q_F(x)\Big)$
5. $\forall x \forall y.\, +1(x,y) \to$
   $\bigvee_{\langle q, inc(c), q'\rangle \in \delta} (q(x) \wedge inc_c(x) \wedge q'(x)) \;\vee\; \bigvee_{\langle q, dec(c), q'\rangle \in \delta} (q(x) \wedge dec_c(x) \wedge q'(x))$
6. $\forall x.\, (\neg \max(x)) \to \bigvee_{c\in C} (inc_c(x) \vee dec_c(x))$
7. $\forall x.\, \bigwedge_{c\in C} \Big(inc_c(x) \to \neg dec_c(x) \wedge \bigwedge_{c'\in C\setminus\{c\}} (\neg dec_{c'}(x) \wedge \neg inc_{c'}(x))\Big)$
8. $\forall x.\, \bigwedge_{c\in C} \Big(dec_c(x) \to \neg inc_c(x) \wedge \bigwedge_{c'\in C\setminus\{c\}} (\neg inc_{c'}(x) \wedge \neg dec_{c'}(x))\Big)$
9. $\forall x.\, (\max(x)) \to \bigwedge_{c\in C} (\neg inc_c(x) \wedge \neg dec_c(x))$
10. $\forall x \forall y. s(x,y) \to \bigvee_{c\in C} (inc_c(x) \wedge dec_c(y))$
11. $\forall x \forall y.\, (s(x,y) \to x < y)$
12. $\forall x.\, \big(\max(x) \vee \exists^{=1} y.\, (s(x,y) \vee s(y,x))\big)$

We will interpret φ_M as a $\mathrm{C}^2[\Sigma_u, \{<, +1, s\}, \{<, +1\}]$ formula. Models of φ_M encode accepting runs of MCA M. The first two conjuncts of φ_M define the meaning of the auxiliary predicates min and max; they hold for the least (resp. the greatest) element of a model. Each element of the model corresponds to precisely one state $q \in Q$, as encoded by conjunct 3. Thus the model is just a sequence of states. The first of them must be the starting state q_I and the last must be a final state $q_F \in F$, as defined by conjunct 4. Every two consecutive elements of the model form a transition. A state in which the transition is fired is marked by predicate of the form inc_c or dec_c denoting a counter to increment or decrement; this is specified by conjunct 5. Every state, with the exception of the last one, must be labelled by precisely one predicate of the form inc_c or dec_c, as expressed by conjuncts 6–8. The last element is not labelled by any of these predicates (conjunct 9), as no transition is fired there. Since values of all counters in starting and final state is 0 and no counter may fall below 0, each incrementation of a counter c must be followed by its decrementation, and conversely, each decrementation of c must be preceded by its incrementation. We use the relation s to match these increments and decrements, as stated in conjunct 10. Conjunct 11 states that decrementation of a counter indeed follows its incrementation. Since each state, except the final one, is a starting state of some transition, it either corresponds to incrementation or decrementation of some counter. Therefore it emits or accepts precisely one edge labelled s, as stated by conjunct 12 of φ_M. Formally, we have the following lemma and a corollary that results from it.

▶ **Lemma 12.** *Let $M = \langle Q, C, \delta, q_I, F\rangle$ be a reduced multicounter automaton and let φ_M be a $\mathrm{C}^2[\Sigma_u, \{<, +1, s\}, \{<, +1\}]$ formula constructed in Definition 11. Formula φ_M is finitely satisfiable if and only if M is non-empty.*

▶ **Corollary 13.** *Finite satisfiability problem for $\mathrm{C}^2[\Sigma_u, \{<, +1, s\}, \{<, +1\}]$ is at least as hard as emptiness problem for multicounter automata.*

6 Satisfiability of $\mathrm{C}^2[\Sigma_u, \Sigma_b, \{<, +1\}]$

In this section we show that the finite satisfiability problem of $\mathrm{C}^2[\Sigma_u, \Sigma_b, \{<, +1\}]$ is decidable.

Fix a finite signature Σ satisfying $\Sigma \subseteq \Sigma_u \cup \Sigma_b$. A *2-type* is a maximal consistent conjunction of atomic and negated atomic formulas over Σ involving only the variables x and y and satisfying three additional restrictions: first, it contains $\neg x = y$; second, whenever it contains $+1(x, y)$ or $+1(y, x)$, it also contains respectively $x < y$ or $y < x$; and third, it contains either $x < y$ or $y < x$, but not both. We will identify a 1-type π (a 2-type τ) with the set of positive atomic formulas occurring in π (in τ). Each 2-type $\tau(x, y)$ uniquely determines two 1-types of x and y, respectively, that we denote $\mathrm{tp}_1(\tau)$ and $\mathrm{tp}_2(\tau)$. For a 2-type τ the 2-type obtained by swapping the variables x and y is denoted τ^{-1}. Symbol $\mathcal{T}(\Sigma)$ denotes the set of of 2-types over Σ.

For a structure $\mathcal{A}$ over the signature Σ and an element $e \in \mathcal{A}$, $\mathrm{tp}^{\mathcal{A}}(e)$ denotes the unique 1-type $\pi \in \Pi(\Sigma)$ such that $\mathcal{A} \models \pi(e)$. Similarly, for $e_1, e_2 \in \mathcal{A}$, $\mathrm{tp}^{\mathcal{A}}(e_1, e_2)$ is the unique 2-type $\tau \in \mathcal{T}(\Sigma)$ such that $\mathcal{A} \models \tau(e_1, e_2)$. If $\mathcal{A} \models \tau(e_1, e_2)$, we say that e_1 *emits* the type τ and e_2 *accepts* it and that τ *originates* in e_1. A 1-type π (resp. 2-type τ) is *realised* in $\mathcal{A}$ if $\pi = \mathrm{tp}^{\mathcal{A}}(e)$ (resp. $\tau = \mathrm{tp}^{\mathcal{A}}(e_1, e_2)$) for some $e \in \mathcal{A}$ (resp. $e_1, e_2 \in \mathcal{A}$, with $e_1 \neq e_2$). Symbols $\Pi(\mathcal{A})$ and $\mathcal{T}(\mathcal{A})$ denote respectively the set of 1-types and the set of 2-types over Σ realised in $\mathcal{A}$. A 1-type $\kappa \in \Pi(\Sigma)$ that has only one realisation in a structure $\mathcal{A}$ is said to be a *king 1-type* in $\mathcal{A}$. If an element e of $\mathcal{A}$ realises a king 1-type then it is said to be a king in $\mathcal{A}$. Any structure may have multiple kings.

If Σ is a relational signature and $\overline{f} = f_1, \ldots, f_m$ is a sequence of distinct binary predicates in Σ, then the pair $\langle \Sigma, \overline{f} \rangle$ is called a *classified signature*. Let $\langle \Sigma, \bar{f} \rangle$ be a classified signature and let $\tau(x, y)$ be a 2-type over Σ. We say that τ is a *message type* over $\langle \Sigma, \bar{f} \rangle$ if $f(x, y) \in \tau(x, y)$ for some distinguished predicate f in $\bar{f}$. Given a structure $\mathcal{A}$ over a signature $\langle \Sigma, \bar{f} \rangle$ and an element $a \in \mathcal{A}$, we want to capture message types connecting a to other elements of $\mathcal{A}$ and all 2-types connecting a to kings of $\mathcal{A}$. We first define the set of all these 2-types. If K is a set of king 1-types from $\mathcal{A}$, then denote by $\tau(K, \Sigma, \bar{f})$ the set of all 2-types μ, such that μ is a message type over $\langle \Sigma, \bar{f} \rangle$ or $\mathrm{tp}_2(\mu) \in K$. A 2-type from $\tau(K, \Sigma, \bar{f})$ is called an *essential type*. If τ is an essential type (resp. a message type) such that τ^{-1} is also an essential type (resp. a message type) then we say that τ is an *invertible essential type* (resp. *invertible message type*). On the other hand, if τ is a 2-type such that neither τ nor τ^{-1} is an essential type, then we say that τ is a *silent type*.

Given a structure $\mathcal{A}$ over a classified signature $\langle \Sigma, \overline{f} \rangle$ and a message type τ, if $\mathcal{A} \models \tau(e_1, e_2)$ then e_2 is called a *witness for* e_1 in $\mathcal{A}$. It follows that if τ is an invertible message type then also e_1 is a witness for e_2. It is because $\mathcal{A} \models \tau^{-1}(e_2, e_1)$ and τ^{-1} is an (invertible) message type.

Since we consider predicates of arity at most 2, a structure $\mathcal{A}$ can be seen as a complete directed graph, where nodes are labelled by 1-types and edges are labelled by 2-types. Thus to define such a structure it is enough to define 1-types of its elements and 2-types of all pairs of elements provided that the projections of 2-types onto 1-types coincide with these 1-types and that for each pair $\langle e_1, e_2 \rangle$ of elements connected by a 2-type μ the pair $\langle e_2, e_1 \rangle$ is connected by the 2-type μ^{-1}.

Normal form of C^2 formulas

For a natural number n denote by $\underline{n}$ the set $\{1, \ldots, n\}$. We will assume that input $\mathrm{C}^2[\Sigma_u, \Sigma_b, \{<, +1\}]$ formula φ is in a normal form

$$\varphi = \quad \forall x \forall y.(\alpha(x, y) \vee x = y) \wedge \quad \bigwedge_{h \in \underline{m}} \forall x \exists^{=1} y.(f_h(x, y) \wedge x \neq y) \tag{1}$$

where α is a quantifier-free formula with unary and binary predicate symbols and $f_1, \ldots, f_m$ are distinguished binary predicates. By a routine adaptation of transformation in [9] we may convert each C^2 formula to an exponentially larger C^2 formula φ' in normal form, such that φ and φ' are equisatisfiable (on structures of cardinality > 1). From now on we also assume that the classified vocabulary of φ is $\langle \Sigma, \bar{f} \rangle$, where $\bar{f} = f_1, \ldots f_m$ and Σ is a finite subset of $\Sigma_u \cup \Sigma_b$.

▶ Remark. In Section 4 we use a notion of normal forms for C^2 formulas with only polynomial blowup. Here, for simplicity of presentation, we decided to employ the one with exponential blowup. Since there is no elementary upper bound on the complexity of the problem to which we reduce our logic, the construction in the present section would not benefit from the usage of a more succinct normal form.

Normal structures

Now, to simplify the reasoning, we restrict the class of models that we consider.

▶ **Definition 14.** A finite structure $\mathcal{A} \in \mathcal{O}(\Sigma, <, \dashv)$ over a classified signature $\langle \Sigma, \bar{f} \rangle$ is *normal* if

1. both the smallest and the largest elements w.r.t. $<^{\mathcal{A}}$ are kings in $\mathcal{A}$,
2. for every two non-king elements $e_1, e_2 \in \mathcal{A}$ satisfying $e_1 <^{\mathcal{A}} e_2$ there exist two elements $e_1', e_2' \in \mathcal{A}$ such that $e_1' <^{\mathcal{A}} e_2'$, $\mathrm{tp}^{\mathcal{A}}(e_1) = \mathrm{tp}^{\mathcal{A}}(e_1')$, $\mathrm{tp}^{\mathcal{A}}(e_2) = \mathrm{tp}^{\mathcal{A}}(e_2')$, and $\mathrm{tp}^{\mathcal{A}}(e_1', e_2')$ is a silent 2-type,
3. for every node $e \in \mathcal{A}$ and $f \in \bar{f}$ we have $|\{e' \in \mathcal{A} \mid \mathcal{A} \models f(e, e')\}| = 1$, and
4. for every $e_1, e_2 \in \mathcal{A}$ if $\dashv^{\mathcal{A}}(e_1, e_2)$ then $\mathrm{tp}^{\mathcal{A}}(e_1, e_2)$ is an invertible essential type.

The following lemma says that when dealing with models of $C^2[\Sigma_u, \Sigma_b, \{<, \dashv\}]$ formulas, we may restrict to normal structures.

▶ **Lemma 15.** *Let φ be a $C^2[\Sigma_u, \Sigma_b, \{<, \dashv\}]$ formula in normal form over a vocabulary $\langle \Sigma, \bar{f} \rangle$. If φ is finitely satisfiable then there exists a vocabulary $\langle \Sigma', \bar{f}' \rangle$ such that $\Sigma \subseteq \Sigma'$ and $\bar{f} \subseteq \bar{f}'$ and a finite normal $\langle \Sigma', \bar{f}' \rangle$-structure $\mathcal{B}$ such that $\mathcal{B} \models \varphi$. Moreover, $|\Sigma'|$ is polynomial in $|\Sigma|$ and $|\bar{f}'| = |\bar{f}| + 2$.*

Star types

Given a structure $\mathcal{A}$ over a signature $\langle \Sigma, \bar{f} \rangle$ and an element $a \in \mathcal{A}$, we want to capture essential 2-types emitted from a to other elements of $\mathcal{A}$. For this reason we introduce star types.

▶ **Definition 16** (Star type in $\mathcal{A}$)**.** Let $\mathcal{A}$ be a normal structure over $\langle \Sigma, \bar{f} \rangle$, and let a be an element of $\mathcal{A}$. Let $K = \{\kappa \mid \kappa \text{ is a king type in } \mathcal{A}\}$. A *star type* of a in $\mathcal{A}$, denoted $\mathrm{st}^{\mathcal{A}}(a)$ is a pair $\sigma = \langle \pi, \mathcal{T} \rangle$ where $\pi = \mathrm{tp}^{\mathcal{A}}(a)$ and $\mathcal{T}$ is the set of essential types originating in a:

$$\mathcal{T} = \{\mu \in \tau(K, \Sigma, \bar{f}) \mid \mathrm{tp}^{\mathcal{A}}(a, b) = \mu \text{ for some } b \in \mathcal{A}\}.$$

We denote the type π by $\pi(\sigma)$. We say that a 2-type μ *occurs* in σ, written $\mu \in \sigma$, if $\mu \in \mathcal{T}$. We write $\sigma - \mu$ for the star-type $\sigma' = \langle \pi, \mathcal{T} \setminus \{\mu\} \rangle$. When S is a set of 2-types we write $\sigma \setminus S$ to denote the star type $\langle \pi, \mathcal{T} \setminus S \rangle$.

Observe that in the definition above σ satisfies the conditions

1. for all $\mu_1, \mu_2 \in \sigma$ if $f(x, y) \in \mu_1$ and $f(x, y) \in \mu_2$ for some $f \in \bar{f}$ then $\mu_1 = \mu_2$,
2. $\mu \in \sigma$ implies $\mathrm{tp}_1(\mu) = \pi$ for all $\mu \in \tau(K, \Sigma, \bar{f})$,

3. $|\{\mu \in \sigma \mid \mathrm{tp}_2(\mu) = \kappa\}| = 1$ for all $\kappa \in K$ such that $\kappa \neq \pi$,
4. $|\{\mu \in \sigma \mid \mathrm{tp}_2(\mu) = \pi\}| = 0$ if $\pi \in K$.

The first of these conditions is obvious, as normal structures emit precisely one edge τ with $f(x,y) \in \tau$ for $f \in \bar{f}$. The second one says that all 2-types originating in a have the same 1-type of the origin, namely the 1-type of a. The third one says that for all kings k (in $\mathcal{A}$) the element a is connected with k by exactly one 2-type (provided that $k \neq a$). The last condition says that if a is a king then it is not connected with itself by any 2-type (recall that 2-types connect different elements).

▶ **Definition 17.** A *star type* over the set of 2-types $\tau(K, \Sigma, \bar{f})$ is any pair of the form $\langle \pi, \mathcal{T} \rangle$ satisfying conditions 1–4 above. A structure $\mathcal{A}$ is said to realise a star type σ if $\mathrm{st}^{\mathcal{A}}(a) = \sigma$ for some $a \in \mathcal{A}$.

For a given set of star types ST, by $\pi(\mathrm{ST})$ we denote the set of 1-types $\{\pi(\sigma) \mid \sigma \in \mathrm{ST}\}$, by $\tau(\mathrm{ST})$ — the set of 2-types occurring in star types from ST, that is the set $\{\mu \mid \exists \sigma \in \mathrm{ST}. \mu \in \sigma\}$, and by *partial*(ST) the set $\{\langle \pi, \mathcal{T}' \rangle \mid \langle \pi, \mathcal{T} \rangle \in \mathrm{ST}$ for some $\mathcal{T}$ satisfying $\mathcal{T}' \subseteq \mathcal{T}\}$. Elements of *partial*(ST) are called *partial star types*. A partial star type is said to be *empty* if it is of the form $\langle \pi, \emptyset \rangle$.

Frames

We now introduce finite and small structures called frames. Frames will be used in deciding the finite satisfiability problems for $\mathrm{C}^2[\Sigma_u, \Sigma_b, \{<, +1\}]$: together with multicounter automata they provide a description of finite models of a given formula.

▶ **Definition 18** (Frame)**.** Let $\langle \Sigma, \bar{f} \rangle$ be a classified signature, K be a set of 1-types over Σ, let ST be a set of star types over $\tau(K, \Sigma, \bar{f})$ and let Ξ be a set of silent 2-types over $\langle \Sigma, \bar{f} \rangle$. A tuple $\langle K, \mathrm{ST}, \Xi, \Sigma, \bar{f} \rangle$ is called *a frame* if the following conditions are satisfied

1. for each 2-type $\tau \in \tau(\mathrm{ST}) \cup \Xi$ if $+1(x,y) \in \tau$ or $+1(y,x) \in \tau$ then τ is an invertible essential type,
2. there exists exactly one star type $\sigma_{first} \in \mathrm{ST}$ such that for every $\tau \in \sigma_{first}$ we have $+1(y,x) \notin \tau$,
3. there exists exactly one star type $\sigma_{last} \in \mathrm{ST}$ such that for every $\tau \in \sigma_{last}$ we have $+1(x,y) \notin \tau$,
4. for each $\kappa \in K$ there exists exactly one $\sigma \in \mathrm{ST}$ such that $\pi(\sigma) = \kappa$, and
5. for each star type $\sigma \in \mathrm{ST}$ and each 2-type μ, if $\mu \in \sigma$ then $\mathrm{tp}_2(\mu) \in \pi(\mathrm{ST})$.

Frames are intended to describe local configurations in normal structures $\mathcal{A}$. The set K contains all king 1-types of $\mathcal{A}$, ST — all star types of $\mathcal{A}$ and the set Ξ — all silent 2-types realised in $\mathcal{A}$. Condition 1 says that every node in $\mathcal{A}$ is connected to its successor and predecessor by invertible essential types. Conditions 2 and 3 say that there are unique star types for the first and the last node in $\mathcal{A}$. Conditions 1–3 follow from the assumption that $\mathcal{A}$ is normal. Condition 4 says that each king has exactly one star type. Condition 5 ensures that if a neighbour of a node in a structure has some 1-type π, then there exists a star type $\sigma \in \mathrm{ST}$ such that $\pi \in \pi(\mathrm{ST})$. The above two conditions hold in every relational structure.

Intuitively, we want to check finite satisfiability of a C^2 formula φ by guessing a right frame. "Right" means here that two conditions must be satisfied. First, the frame should be locally consistent with φ. This means that every 2-type occurring in the frame entails the subformulas of φ of the form $\forall x \forall y \ldots$, and that the number of witnesses in every star type is correct. This is formalised in the following definition. Second, the frame should be

globally consistent in the sense that there exists a structure that conforms to this frame — this is formalised in Definition 20.

▶ **Definition 19** ($\mathcal{F} \models \varphi$)**.** Consider a frame $\mathcal{F} = \langle K, \mathrm{ST}, \Xi, \Sigma, \bar{f}\rangle$ and a $\mathrm{C}^2[\Sigma_u, \Sigma_b, \{<, +1\}]$ formula φ in normal form (1) over $\langle \Sigma, \bar{f}\rangle$. We say that $\mathcal{F}$ *satisfies* φ, in symbols $\mathcal{F} \models \varphi$, if

- for each 2-type $\mu \in \Xi \cup \tau(\mathrm{ST})$, the formula α is a consequence of μ and of μ^{-1}, that is $\models \mu \to \alpha$ and $\models \mu^{-1} \to \alpha$, where μ is seen as conjunction of literals, and
- for each $\sigma \in \mathrm{ST}$ and $h \in \underline{m}$ we have $|\{\mu \in \sigma \mid f_h(x, y) \in \mu\}| = 1$.

▶ **Definition 20.** Let $\langle K, \mathrm{ST}, \Xi, \Sigma, \bar{f}, c\rangle$ be a frame, and $\mathcal{A}$ structure over $\langle \Sigma, \bar{f}\rangle$. We say that $\mathcal{A}$ *fits to the frame* $\mathcal{F}$ if

- the set of king 1-types realised in structure $\mathcal{A}$ is K, and
- the set of all silent types realised in $\mathcal{A}$ is a subset of Ξ, and
- the set of star types of $\mathcal{A}$ is a subset of ST, in symbols $\mathrm{st}^{\mathcal{A}}(\mathcal{A}) \subseteq \mathrm{ST}$.

The following proposition reduces the finite satisfiability problem of $\mathrm{C}^2[\Sigma_u, \Sigma_b, \{<, +1\}]$ to the problem of existence of a structure in $\mathcal{O}(\Sigma, <, +1)$ that fits to a given frame.

▶ **Proposition 21.** *Let φ be a $\mathrm{C}^2[\Sigma_u, \Sigma_b, \{<, +1\}]$ formula in normal form over a vocabulary $\langle \Sigma, \bar{f}\rangle$, where $\{<, +1\} \subseteq \Sigma$. Let $\mathcal{A}$ be a structure in $\mathcal{O}(\Sigma, <, +1)$.*

1. *If $\mathcal{A}$ is normal and $\mathcal{A} \models \varphi$ then there exists a frame $\mathcal{F}$, such that $\mathcal{A}$ fits to $\mathcal{F}$ and $\mathcal{F} \models \varphi$.*
2. *If there exists a frame $\mathcal{F}$ such that $\mathcal{A}$ fits to $\mathcal{F}$ and $\mathcal{F} \models \varphi$ then $\mathcal{A} \models \varphi$.*

Proof. For the proof of the first statement, assume that $\mathcal{A}$ is normal and $\mathcal{A} \models \varphi$. Let K be the set of king 1-types realised in $\mathcal{A}$, let ST be the set of star types of $\mathcal{A}$ and let Ξ be the set of all silent types realised in $\mathcal{A}$. The facts that tuple $\mathcal{F} = \langle K, \mathrm{ST}, \Xi, \Sigma, \bar{f}, c\rangle$ forms a frame, $\mathcal{F} \models \varphi$ and $\mathcal{A}$ fits to $\mathcal{F}$ are immediate, once Definitions 18, 19 and 20 are spelled.

For the proof of the second statement, let $\mathcal{F}$ be a frame such that $\mathcal{F} \models \varphi$ and let $\mathcal{A}$ be a structure such that $\mathcal{A}$ fits to $\mathcal{F}$. Since φ is in normal form, it is of the form (1). Let μ be any 2-type realised in $\mathcal{A}$. Then either μ is a silent type or it occurs in some star type realised in $\mathcal{A}$. In any case, by Definition 20 we have that $\mu \in \Xi \cup \tau(\mathrm{ST})$. By Definition 19 it follows that $\models \mu \to \alpha$. So $\mathcal{A} \models \forall x \forall y.(\alpha \vee x = y)$. Since $\mathcal{A}$ fits to $\mathcal{F}$ and $\mathcal{F} \models \varphi$, it also follows that for each star type σ realised in $\mathcal{A}$ and each h such that $1 \leq h \leq m$ we have $|\{\mu \in \sigma \mid f_h(x, y) \in \mu\}| = 1$, and thus $\mathcal{A} \models \bigwedge_{h \in \underline{m}} \forall x \exists^{=1} y.(f_h(x, y) \wedge x \neq y)$. Hence $\mathcal{A} \models \varphi$ as required. ◀

High-level multicounter automata

In the rest of this section we will want to decide for a given frame $\mathcal{F}$ if there exists a structure in $\mathcal{O}(\Sigma, <, +1)$ that fits to this frame. This will be done by a reduction to emptiness problem for Multicounter Automata.

We now introduce a syntactic extension to multicounter automata that we call *High-level MultiCounter Automata* (**HMCA**). The idea is to specify transitions of an automaton as programs in a higher-level imperative language with conditionals, loops and arrays, which leads to clearer exposition of reachability problems. A transition in a High-Level MCA is a sequence Δ of actions; each action in turn may update and test finite-domain variables, and conduct conditional or loop instructions depending on results of these tests. A transition may also increment or decrement, but not test the value of, counters, which are the only infinite-domain variables of the automaton.

Formally, an HMCA is a tuple $H = \langle \mathrm{V}_{fin}, \mathrm{Vec}_{\mathbb{N}}, \mathrm{Type}, \Delta, \rho_I, P_F, E\rangle$. Set V_{fin} consists of variables v to be interpreted in the finite domain $\mathrm{Type}(v)$. We may think of V_{fin} as a declaration of finite-domain variables of the program. Set $\mathrm{Vec}_{\mathbb{N}}$ corresponds to a declaration

of arrays, it consists of variables $\overrightarrow{vec}$ to be interpreted as vectors of natural numbers indexed by elements of some finite set $\mathbb{A}$, where $\mathrm{Type}(\overrightarrow{vec}) = \mathbb{A} \to \mathbb{N}$. We will refer to the index set $\mathbb{A}$ as the domain $\mathrm{Dom}(\overrightarrow{vec})$. The Type function assigns to every variable in $\mathrm{V}_{fin} \cup \mathrm{Vec}_{\mathbb{N}}$ its type. The sequence of actions Δ is the actual program built from actions defined below. The starting state of H is ρ_I, the set of accepting states is P_F. Set E is a subset of $\{\langle \vec{v}, a\rangle \mid \vec{v} \in \mathrm{Vec}_{\mathbb{N}},\ a \in \mathrm{Dom}(\vec{v})\}$, and is used in the acceptance condition explained later.

We now define a set of actions α that constitute transitions in **HMCA**. The simplest action is an *assignment* of the form $v := \mathit{Expr}$, where v is a variable of some domain $\mathbb{A} = \mathrm{Type}(v)$, and *Expr* is an expression built from variables from V_{fin}, constants from appropriate domains and operators. An *operator* is any effectively computable function, e. g., $\cup, \cap, \setminus$ are operators of domain $(2^{\mathbb{B}})^2 \to 2^{\mathbb{B}}$ for any domain $\mathbb{B}$; function $\pi : \mathrm{ST}(\mathcal{K}, \Sigma, \bar{f}) \to \Pi(\Sigma)$ from Definition 16 is also an operator, provided that our finite domain contains $\mathrm{ST}(\mathcal{K}, \Sigma, \bar{f})$ and $\Pi(\Sigma)$. We silently extend Type function to constants by letting $\mathrm{Type}(a) = \mathbb{A}$ if $a \in \mathbb{A}$, and to expressions, e. g., $\mathrm{Type}(s_1 \cup s_2) = 2^{\mathbb{B}}$ if $\mathrm{Type}(s_1) = 2^{\mathbb{B}}$ and $\mathrm{Type}(s_2) = 2^{\mathbb{B}}$. We require that assignments $v := \mathit{Expr}$ are well typed, i. e., that $\mathrm{Type}(v) = \mathrm{Type}(\mathit{Expr})$. An *atomic test* is of the form $\mathit{Expr}_1 = \mathit{Expr}_2$ or $\mathit{Expr}_3 \in \mathit{Expr}_4$, where Expr_1, Expr_2, Expr_3, Expr_4 are expressions such that $\mathrm{Type}(\mathit{Expr}_1) = \mathrm{Type}(\mathit{Expr}_2)$ and $\mathrm{Type}(\mathit{Expr}_4) = 2^{\mathrm{Type}(\mathit{Expr}_3)}$. A *test* is an arbitrary Boolean combination of *atomic tests*. Notice that tests do not use counters. A *non-deterministic assignment* action is of the form **guess** $v \in \mathit{Expr}$ **with** *Test*, where $\mathrm{Type}(\mathit{Expr}) = 2^{\mathrm{Type}(v)}$ and the variable v may occur in *Test*. A conditional action is of the form **if** *Test* **then** α^* **else** α'^* **endif** or **if** *Test* **then** α^* **endif**. A loop action is of the form **while** *Test* **do** α^* **endwhile**. An *incrementing action* (resp. *decrementing action*) is of the form $\mathbf{inc}(\vec{f}[\mathit{Expr}])$ (resp. $\mathbf{dec}(\vec{f}[\mathit{Expr}])$), where *Expr* evaluates to an index of the array $\vec{f}$, that is, $\vec{f} \in \mathrm{Vec}_{\mathbb{N}}$, $\mathrm{Type}(\vec{f}) = \mathbb{A} \to \mathbb{N}$ and $\mathrm{Type}(\mathit{Expr}) = \mathbb{A}$. The remaining action, **Reject**, simply rejects current computation.

Expressions and tests are evaluated in context of variable valuations. A *variable valuation* (also called a *state*) is any function ρ that assigns to every finite-domain variable v a value $[\![v]\!]_\rho \in \mathrm{Type}(v)$.We write $[\![v]\!]_\rho = \rho(v)$ for $v \in \mathbb{V}_{fin}$, $[\![\mathit{Expr}_1 \bowtie \mathit{Expr}_2]\!]_\rho = [\![\mathit{Expr}_1]\!]_\rho \bowtie [\![\mathit{Expr}_2]\!]_\rho$, where $\bowtie \in \{\cup, \cap, \setminus\}$ and $[\![f(\mathit{Expr}_1, \ldots, \mathit{Expr}_k)]\!]_\rho = f([\![\mathit{Expr}_1]\!]_\rho, \ldots, [\![\mathit{Expr}_k]\!]_\rho)$, where f is an operator. In a similar way we define semantics of tests.

A *counter valuation* is any function ϑ that assigns (a sequence of) natural numbers to (arrays of) counters. A *configuration* of **HMCA** H is a pair $\langle \rho, \vartheta\rangle$ where ρ is an variable valuation (i. e., a state) and ϑ is a counter valuation. Actions transform configurations. Most actions work only on variable valuations; the exceptions are incrementing and decrementing of counters. With the exception of the decrementing action, the semantics of actions is self-explanatory; $\mathit{dec}(c)$ decrements the counter c if it is strictly positive and otherwise (if it is 0) it rejects the current computation.

A *run* of an HMCA H is a sequence of configurations $\langle \rho_1, \vartheta_1\rangle, \ldots \langle \rho_k, \vartheta_k\rangle$ such that $\langle \rho_{i+1}, \vartheta_{i+1}\rangle$ is obtained after executing transition Δ in configuration $\langle \rho_i, \vartheta_i\rangle$, for $i \in \{1, \ldots, k-1\}$. A run is *accepting*, if it starts in an initial configuration $\langle \rho_I, \vartheta_0\rangle$ with ρ_I being initial state and ϑ_0 assigning 0 to all counters, and it ends in some configuration $\langle \rho_F, \vartheta_F\rangle$ with ρ_F being a final state and ϑ_F assigning 0 to all counters specified in the set E of final counters: $\vartheta_F(\vec{f})(a) = 0$ for every $\langle f, a\rangle \in E$. The emptiness problem for high-level multicounter automata is the question whether a given automaton H has an accepting run.

In the full version of the paper we give formal syntax and semantics to high-level multicounter automata and we prove that HMCA can be compiled to multicounter automata. Intuitively, the control structures and finite-domain variables (including tests for zero on finite-domain variables) can be hidden in states of the constructed MCA. Formally, we have the following proposition.

▶ **Proposition 22.** *Emptiness problem for* ***HMCA*** *is reducible to emptiness problem of multicounter automata, and is therefore decidable.*

Figure 1 shows a high-level multicounter automaton $H_{\mathcal{F}}$ that for a given frame $\mathcal{F}$ checks whether there exists a normal structure that fits to $\mathcal{F}$. The automaton guesses one by one the sequence of elements of the structure as they appear in the order $<$. The constructed HMCA keeps track of the set of nodes visited (i. e., guessed) so far; their 1-types are stored in variable Visited; variable Required stores the set of king types that still must be constructed. These two variables are updates in lines 4–7.

A crucial notion in the construction of the automaton is the difference type of a node w.r.t. to another node in a structure. Fig 2 shows an example of a difference type.

▶ **Definition 23** (Difference type of e_1 w.r.t. e in a structure $\mathcal{A}$)**.** Let $\mathcal{A} \in \mathcal{O}(\Sigma, <, \dashv\!\!\!\perp)$ and $e_1, e \in \mathcal{A}$ be elements satisfying $e_1 <^{\mathcal{A}} e$. Let σ be the star type of e_1 and let $\{\tau_i\}_{i=1}^{k}$ be essential types emitted from e_1 and accepted by nodes $<^{\mathcal{A}}$ than e. A partial star type $\sigma \setminus \{\tau_i\}_{i=1}^{k}$ is called a *difference type of* e_1 *w.r.t.* e *in* $\mathcal{A}$.

▶ **Definition 24.** Let $A \in \mathcal{O}(\Sigma, <, \dashv\!\!\!\perp)$ and $e \in \mathcal{A}$. The *cut* at point e in $\mathcal{A}$ is a vector $\overrightarrow{\mathsf{Cut}}_e$ of natural numbers indexed by star types on $\tau(K, \Sigma, \bar{f})$ such that

$$\overrightarrow{\mathsf{Cut}}_e[\sigma] = |\{e_1 \in \mathcal{A} \mid \text{difference type of } e_1 \text{ w.r.t. } e \text{ in } \mathcal{A} \text{ is } \sigma\}|.$$

Intuitively, the cut vector $\overrightarrow{\mathsf{Cut}}_e$ informs us, for each 2-type τ, how many edges of type τ emitted by nodes smaller than e must be accepted by nodes greater or equal to e. Additionally, it informs which of these edges have common origin, i. e., they belong to a star type of the same node. This information is used when we define 2-types connecting e with smaller nodes. When we define a 2-type connecting e with a node e' smaller than e, we have to subtract this 2-type from the current cut. At the same time we have to remember that for each pair of nodes there is only one 2-type connecting them, so when we subtract two 2-types from a cut, we have to be sure that their origins are different. This is why the cut vector is indexed by difference types and not by 2-types.

For a star type σ define $\sigma^{<} = \{\tau \in \sigma \mid (y < x) \in \tau\}$ and $\sigma^{>} = \{\tau \in \sigma \mid (x < y) \in \tau\}$. Intuitively $\sigma^{<}$ (resp. $\sigma^{>}$) denotes the subset of σ containing essential types emitted to smaller (resp. larger) nodes. For a partial star type σ from ST define $\tau_{\dashv\!\!\!\perp}(\sigma)$ as the only 2-type τ such that $\dashv\!\!\!\perp(x, y) \in \tau$, or the special value $\perp$ if σ is the star type of last node of a structure. Similarly, define $\mathit{first}(\sigma)$ to be an arbitrary 2-type τ such that $\tau \in \sigma$. The value of $\overrightarrow{\mathsf{Cut}}$ is updated in two loops in lines 9–27. During the computation some counters from $\overrightarrow{\mathsf{Cut}}$ are decremented, and some counters from $\overrightarrow{\mathsf{Processed}}$ — incremented. Decrementation of a counter corresponds to establishing a 2-type between e and some e' smaller than e (this is done in the loop in lines 9–19), or between e' and e (loop in lines 21–27). In order not to establish multiple 2-types between the same pair of nodes, we remove the difference type of e' w.r.t. e from $\overrightarrow{\mathsf{Cut}}$ and store it in $\overrightarrow{\mathsf{Processed}}$. When the second loop (lines 21–27) finishes its execution the initial value of $\overrightarrow{\mathsf{Cut}}$ vector for next node is the sum of the values of updated vectors $\overrightarrow{\mathsf{Cut}}$ and $\overrightarrow{\mathsf{Processed}}$ in line 27.

In lines 29–34 we guess the star type of the next node, or the special value $\perp$ in case the maximal element of the structure is already guessed. To keep the constructed structure normal (and to be able to define silent 2-types between non-king nodes) we have to satisfy condition 2 in Definition 14. Therefore, while guessing a consecutive node, we must check that it does not violate this condition. Therefore we guess (the star type of) the consecutive

Type $\mathrm{Type}(\sigma_c) = \mathrm{ST} \cup \{\bot\}$, $\mathrm{Type}(\sigma)$, $\mathrm{Type}(\sigma_u)$ and $\mathrm{Type}(\sigma_g)$ is *partial*(ST), $\mathrm{Type}(\tau) = \tau(\mathrm{ST})$ and $\mathrm{Type}(\tau_{\text{Ш}}) = \tau(\mathrm{ST}) \cup \{\bot\}$. $\mathrm{Type}(\text{Required}) = 2^K$ and $\mathrm{Type}(\text{Visited}) = 2^{\pi(ST)}$. $\mathrm{Type}(\overrightarrow{\mathsf{Cut}})$ and $\mathrm{Type}(\overrightarrow{\mathsf{Processed}})$ is *partial*(ST) $\to \mathbb{N}$.

Initial Configuration Initial state is ρ_I such that $\rho_I(\sigma_c) = \sigma_{first}$, $\rho_I(\text{Required}) = K$, $\rho_I(\text{Visited}) = \emptyset$ and the value of ρ_I on remaining variables is arbitrary (but fixed). Initial counter valuation assigns 0 to all counters.

Accepting Configurations The set of accepting states P_F consists of all states ρ_F satisfying $K \subseteq \rho_F(\text{Visited})$. Accepting counter valuations are defined by the set $E = \{\langle \overrightarrow{\mathsf{Cut}}, \sigma \rangle \mid \sigma \in \textit{partial}(\mathrm{ST}) \text{ is non-empty}\}$.

Transition Δ

if $\sigma_c = \bot$ **then**
  Reject
endif
if $\pi(\sigma_c) \in$ Required **then**
  Required := Required $\setminus \{\pi(\sigma_c)\}$
endif
Visited := Visited $\cup \{\pi(\sigma_c)\}$
$\sigma := {\sigma_c}^<$
while $\sigma \neq \emptyset$ **do**
  $\tau := \textit{first}(\sigma)$
  $\sigma := \sigma - \tau$
  if τ is an invertible essential type **then**
    guess $\sigma_u \in \textit{partial}(\mathrm{ST})$ **with** $\tau^{-1} \in {\sigma_u}^>$
  else
    guess $\sigma_u \in \textit{partial}(\mathrm{ST})$ **with** $\pi(\sigma_u) = \mathrm{tp}_2(\tau)$
  endif
  dec($\overrightarrow{\mathsf{Cut}}[{\sigma_u}^>]$)
  inc($\overrightarrow{\mathsf{Processed}}[{\sigma_u}^> - \tau^{-1}]$)
endwhile
guess *anotherIteration* $\in \{$**true**, **false**$\}$
while *anotherIteration* **do**
  guess $\sigma_g \in \textit{partial}(\mathrm{ST})$
  guess $\tau \in {\sigma_g}^>$ **with** $\mathrm{tp}_2(\tau) = \pi(\sigma_c)$ **and** (τ is non-invertible essential type)
  dec($\overrightarrow{\mathsf{Cut}}[{\sigma_g}^>]$)
  inc($\overrightarrow{\mathsf{Processed}}[{\sigma_g}^> - \tau]$)
  guess *anotherIteration* $\in \{$**true**, **false**$\}$
endwhile
inc($\overrightarrow{\mathsf{Cut}}[{\sigma_c}^>]$)
$\tau_{\text{Ш}} := \tau_{\text{Ш}}({\sigma_c}^>)$
if $\tau_{\text{Ш}} = \bot$ **then**
  $\sigma_c := \bot$
else
  guess $\sigma_c \in$ Allowed(Visited) **with** $(\tau_{\text{Ш}})^{-1} \in \sigma_c$
endif
guess *anotherIteration* $\in \{$**true**, **false**$\}$
while *anotherIteration* **do**
  guess $\sigma_g \in \textit{partial}(\mathrm{ST})$
  dec($\overrightarrow{\mathsf{Processed}}[{\sigma_g}^>]$)
  inc($\overrightarrow{\mathsf{Cut}}[{\sigma_g}^>]$)
  guess *anotherIteration* $\in \{$**true**, **false**$\}$
endwhile

Figure 1 A high-level multicounter automaton $H_{\mathcal{F}}$ corresponding to a frame $\mathcal{F}$. Here the set of finite-domain variables is $\{\sigma_c, \text{Visited}, \text{Required}, \sigma, \tau, \sigma_u, \sigma_g, \tau_{\text{Ш}}, \textit{anotherIteration}\}$, and there are two arrays of counters $\overrightarrow{\mathsf{Cut}}$ and $\overrightarrow{\mathsf{Processed}}$.

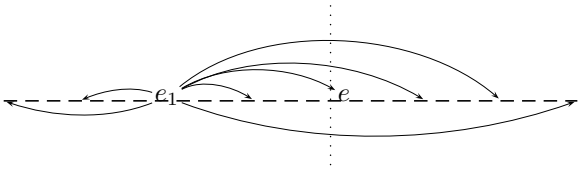


Figure 2 Difference type of e_1 w.r.t. e contains the arrows that cross the dotted line.

node from the set

$$\begin{aligned}&\text{Allowed}(\text{Visited}) = \{\sigma \in \text{ST} \mid \pi(\sigma) \in K \setminus \text{Visited}\} \cup \\ &\quad \{\sigma \in \text{ST} \mid \forall \pi \in \text{Visited}\,(\pi \notin K \Rightarrow \exists \tau \in \Xi.\text{tp}_1(\tau) = \pi \wedge (x < y) \in \tau \wedge \text{tp}_2(\tau) = \pi(\sigma))\}.\end{aligned}$$

Finally, loop in lines 35–41 may move the content of vector $\overrightarrow{\mathsf{Processed}}$ to $\overrightarrow{\mathsf{Cut}}$. We may assume that the entire content is actually moved. Formally, the correspondence between a frame $\mathcal{F}$ and HMCA $H_{\mathcal{F}}$ is captured by the following proposition, which directly leads to the main theorem of this section.

▶ **Proposition 25.** *Let $\mathcal{F} = \langle K, \text{ST}, \Xi, \Sigma, \bar{f}\rangle$ be a frame. The automaton $H_{\mathcal{F}}$ is non-empty if and only if there exists a structure $\mathcal{M} \in \mathcal{O}(\Sigma, <, +1)$ that fits to $\mathcal{F}$.*

▶ **Theorem 26.** *The finite satisfiability problem for $\text{C}^2[\Sigma_u, \Sigma_b, \{<, +1\}]$ is decidable.*

Proof. We may assume that the input $\text{C}^2[\Sigma_u, \Sigma_b, \{<, +1\}]$ formula φ is in normal form (otherwise it can be brought to the normal form). A non-deterministic decision procedure for the finite satisfiability problem guesses a frame $\mathcal{F}$ such that $\mathcal{F} \models \varphi$ and checks if HMCA $H_{\mathcal{F}}$ is non-empty. If so, then by Proposition 25 we obtain a structure $\mathcal{M} \in \mathcal{O}(\Sigma, <, +1)$ that fits to $\mathcal{F}$. Because $\mathcal{M}$ fits to $\mathcal{F}$ and $\mathcal{F} \models \varphi$, by Proposition 21 we conclude that $\mathcal{M} \models \varphi$. This shows that φ is finitely satisfiable, so our procedure is sound. On the other hand, if φ is finitely satisfiable then, by Lemma 15, it has a model $\mathcal{M}$ which is a normal structure. Again, by Proposition 21 there exists a frame $\mathcal{F}$ such that $\mathcal{M}$ fits to $\mathcal{F}$ and $\mathcal{F} \models \varphi$. By Proposition 25 we conclude that $H_{\mathcal{F}}$ is non-empty, so the procedure is complete.

Note that the size of $\mathcal{F}$ is at most doubly exponential in the size of formula's vocabulary $\langle \Sigma, \bar{f}\rangle$, so there are finitely many frames that can be guessed, and that the emptiness problem of HMCA $H_{\mathcal{F}}$ is decidable, as stated in Proposition 22. ◀

7 Conclusion

We have shown several complexity results for finite satisfiability of two-variable logics with counting quantifiers and linear orders. In particular we proved NExpTime-completeness of the problem for $\text{C}^2[\Sigma_u, \{<, +1\}, \{<, +1\}]$, VAS-completeness for $\text{C}^2[\Sigma_u, \Sigma_b, \{<\}]$ and undecidability for $\text{C}^2[\Sigma_u, \{<_1, s_1, <_2, s_2\}, \{<_1, <_2\}]$. There are still some unsolved cases, including $\text{C}^2[\Sigma_u, \{<_1, <_2\}, \{<_1, <_2\}]$ and $\text{C}^2[\Sigma_u, \{<_1, s_1, <_2\}, \{<_1, <_2\}]$.

There are lots of open problems in the area. One of them is general satisfiability. None of the logics considered here has finite model property. Our techniques rely on finiteness of the underlying structure, so they cannot be directly applied to general satisfiability on possibly infinite structures. Among possible directions for future work one can choose combination of C^2 with other interpreted binary relations like preorders [21] or transitive relations [33]. Another possibility is to consider C^2 with closure operations on some relations, like equivalence closure [14] or deterministic transitive closure [3].

References

1 Mikolaj Bojanczyk, Anca Muscholl, Thomas Schwentick, Luc Segoufin, and Claire David. Two-variable logic on words with data. In *LICS*, pages 7–16. IEEE Computer Society, 2006.

2 Witold Charatonik, Emanuel Kieronski, and Filip Mazowiecki. Satisfiability of the two-variable fragment of first-order logic over trees. *CoRR*, abs/1304.7204, 2013.

3 Witold Charatonik, Emanuel Kieroński, and Filip Mazowiecki. Decidability of weak logics with deterministic transitive closure. In *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS'14, Vienna, Austria, July 14–18, 2014*, page 29, 2014.

4 Witold Charatonik and Piotr Witkowski. Two-variable logic with counting and trees. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*, pages 73–82, 2013.

5 Kousha Etessami, Moshe Y. Vardi, and Thomas Wilke. First-order logic with two variables and unary temporal logic. *Inf. Comput.*, 179(2):279–295, 2002.

6 Diego Figueira. Satisfiability for two-variable logic with two successor relations on finite linear orders. *Computing Research Repository*, abs/1204.2495, 2012.

7 E. Grädel, M. Otto, and E. Rosen. Undecidability results on two-variable logics. *Archive for Mathematical Logic*, 38:213–354, 1999.

8 Erich Grädel, Phokion Kolaitis, and Moshe Vardi. On the decision problem for two-variable first-order logic. *Bulletin of Symbolic Logic*, 3(1):53–69, 1997.

9 Erich Grädel and Martin Otto. On logics with two variables. *Theor. Comput. Sci.*, 224(1-2):73–113, 1999.

10 Erich Grädel, Martin Otto, and Eric Rosen. Two-variable logic with counting is decidable. In *LICS*, pages 306–317. IEEE Computer Society, 1997.

11 N. Immerman, A. Rabinovich, T. Reps, M.Sagiv, and G. Yorsh. The boundary between decidability and undecidability for transitive-closure logics. In *Proceedings of the 18th Annual Conference of the European Association for Computer Science Logic (CSL'04)*, pages 160–174, 2004.

12 Emanuel Kieroński. Decidability issues for two-variable logics with several linear orders. In Marc Bezem, editor, *CSL*, volume 12 of *LIPIcs*, pages 337–351. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2011.

13 Emanuel Kieroński and Jakub Michaliszyn. Two-variable universal logic with transitive closure. In Patrick Cégielski and Arnaud Durand, editors, *CSL*, volume 16 of *LIPIcs*, pages 396–410. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012.

14 Emanuel Kieroński, Jakub Michaliszyn, Ian Pratt-Hartmann, and Lidia Tendera. Two-variable first-order logic with equivalence closure. In *LICS*, pages 431–440. IEEE, 2012.

15 Emanuel Kieroński and Martin Otto. Small substructures and decidability issues for first-order logic with two variables. In *LICS*, pages 448–457. IEEE Computer Society, 2005.

16 Emanuel Kieroński and Lidia Tendera. On finite satisfiability of two-variable first-order logic with equivalence relations. In *LICS*, pages 123–132. IEEE Computer Society, 2009.

17 S. Rao Kosaraju. Decidability of reachability in vector addition systems (preliminary version). In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, pages 267–281, 1982.

18 Jérôme Leroux. The general vector addition system reachability problem by presburger inductive invariants. In *Proceedings of the 24th Annual IEEE Symposium on Logic in Computer Science*, pages 4–13, 2009.

19 R. J. Lipton. The reachability problem requires exponential space. 62, New Haven, Connecticut: Yale University, Department of Computer Science, Research, Jan, 1976.

20 Amaldev Manuel. Two variables and two successors. In Petr Hlinený and Antonín Kucera, editors, *MFCS*, volume 6281 of *Lecture Notes in Computer Science*, pages 513–524. Springer, 2010.

21 Amaldev Manuel and Thomas Zeume. Two-variable logic on 2-dimensional structures. In Simona Ronchi Della Rocca, editor, *Computer Science Logic 2013 (CSL 2013), CSL 2013, September 2-5, 2013, Torino, Italy*, volume 23 of *LIPIcs*, pages 484–499. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2013.

22 Ernst W. Mayr. An algorithm for the general Petri net reachability problem. *SIAM J. Comput.*, 13(3):441–460, 1984.

23 Michael Mortimer. On languages with two variables. *Mathematical Logic Quarterly*, 21(1):135–140, 1975.

24 B. O. Nash. Reachability problems in vector addition systems. *The American Mathematical Monthly*, 80(3):pp. 292–295, 1973.

25 Martin Otto. Two variable first-order logic over ordered domains. *J. Symb. Log.*, 66(2):685–702, 2001.

26 Leszek Pacholski, Wieslaw Szwast, and Lidia Tendera. Complexity of two-variable logic with counting. In *LICS*, pages 318–327. IEEE, 1997.

27 Ian Pratt-Hartmann. Complexity of the two-variable fragment with counting quantifiers. *Journal of Logic, Language and Information*, 14(3):369–395, 2005.

28 Ian Pratt-Hartmann. The two-variable fragment with counting revisited. In Anuj Dawar and Ruy J. G. B. de Queiroz, editors, *WoLLIC*, volume 6188 of *Lecture Notes in Computer Science*, pages 42–54. Springer, 2010.

29 Ian Pratt-Hartmann. Logics with counting and equivalence. In *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS'14, Vienna, Austria, July 14–18, 2014*, page 76, 2014.

30 Thomas Schwentick and Thomas Zeume. Two-variable logic with two order relations – (extended abstract). In Anuj Dawar and Helmut Veith, editors, *CSL*, volume 6247 of *Lecture Notes in Computer Science*, pages 499–513. Springer, 2010.

31 Dana Scott. A decision method for validity of sentences in two variables. *Journal of Symbolic Logic*, 27:477, 1962.

32 Luc Segoufin. Automata and logics for words and trees over an infinite alphabet. In *Proceedings of CSL 2006*, volume 4207 of *Lecture Notes in Computer Science*, pages 41–57. Springer, 2006.

33 Wieslaw Szwast and Lidia Tendera. FO^2 with one transitive relation is decidable. In *30th International Symposium on Theoretical Aspects of Computer Science (STACS 2013)*, volume 20 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 317–328, Dagstuhl, Germany, 2013. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

Binding Forms in First-Order Logic

Fabio Mogavero[*1,2] **and Giuseppe Perelli**[†1]

1 University of Oxford, U.K.
2 Università degli Studi di Napoli Federico II, Italy

Abstract

Aiming to pinpoint the reasons behind the decidability of some complex extensions of *modal logic*, we propose a new *classification criterion* for sentences of *first-order logic*, which is based on the kind of *binding forms* admitted in their expressions, *i.e.*, on the way the arguments of a relation can be bound to a variable. In particular, we describe a hierarchy of four fragments focused on the Boolean combinations of these forms, showing that the less expressive one is already incomparable with several first-order restrictions proposed in the literature, as the *guarded* and *unary negation* fragments. We also prove, via a novel model-theoretic technique, that our logic enjoys the finite-model property, Craig's interpolation, and Beth's definability. Furthermore, the associated model-checking and satisfiability problems are solvable in PTime and Σ_3^{P}, respectively.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases First-Order Logic, Decidable Fragments, Satisfiability, Model Checking

Digital Object Identifier 10.4230/LIPIcs.CSL.2015.648

1 Introduction

Since from the revolutionary negative solutions owed to Church and Turing, Hilbert's original "*Entscheidungsproblem*" [27] turned into a vast *classification process* looking for all those classes of *first-order sentences* having a *decidable satisfiability* [10]. Depending upon the syntactic criteria used to identify the particular classes of interest [23], this process was declined into several research programs, among which we can mention, on one side, those limiting *relation arities* [35] or the total *number of variables* [40, 24] and, on the other one, those classifying sentences in *prenex normal form* based on their *prefix vocabulary* [11].

The research in this field has had a considerable impact, from both a theoretical and practical point of view, in a variety of areas on the edge between mathematics and computer science, *e.g.*, *reverse mathematics* [48], *descriptive complexity* [33], *database theory* [51, 1], and *formal verification*, just to mention a few. However, Vardi observed that almost all of the classic approaches did not shed any satisfactory light on why *modal logic* and derived frameworks, like the ones featuring fixpoint constructs, are so robustly decidable [57, 21].

Trying to find a plausible answer, Andréka, van Benthem, and Németi introduced the *guarded fragment* of first-order logic [3], which generalizes the modal framework by essentially retaining several of its model-theoretic and algorithmic properties. This work started a completely new research program based on the way quantifications can be *relativised* to atoms, avoiding the usual syntactic restrictions on quantifier patterns, number of variables, and relation arities. Pushing forward the idea that robust fragments of first-order logic

* Research supported by the INdAM-GNCS Funding "Giovani Ricercatori 2014".
† Work partially done when the author was a Ph.D. student at the Università degli Studi di Napoli Federico II. Research partially supported by the ERC Advanced Grant RACE (291528) at Oxford.

24th EACSL Annual Conference on Computer Science Logic (CSL 2015).
Editor: Stephan Kreutzer; pp. 648–665

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

owe their nice properties to some sort of guarded quantification, several extensions along this line of research were proposed in the literature, such as the *loosely guarded* [52], the *clique guarded* [19, 22], the *action guarded* [53, 18], and the *guarded fixpoint logic* [25]. This classification program has also important applications in database theory and description logic, where it is relevant to evaluate a query against guarded first-order theories [5].

Only recently, ten Cate and Segoufin observed that the first-order translation of modal logic presents, besides the guarded nature of quantifications, another important peculiarity: negation is only applied to sentences or monadic formulas, *i.e.*, formulas with a single free variable. Exploiting this observation, they introduced a new robust fragment of first-order logic, called *unary negation* [49, 50], which extends modal logic, as well as other formalisms, like Boolean conjunctive queries, that cannot be expressed in terms of guarded quantifications. Since this new restriction is incomparable with the guarded fragments, right after the original work, another formalism was proposed, called *guarded negation* [7], which unifies the two approaches. Syntactically, there is no primary universal quantifier and the use of negation is only allowed if guarded by an atom. In terms of expressive power, this fragment forms a strict extension of both the logics on which it is based, while preserving the same desirable properties of the modal framework. However, it has to be noted that it is still incomparable with more complex extensions of the guarded fragment, such as the clique guarded one. This way of analysing formulas focusing on the guarded nature of negation has also important applications to database theory, where it is well-known that the operation of complementation makes queries hard to evaluate [6].

Although these two innovative classification programs really succeeded in the original task to explain the nice properties enjoyed by modal logic, we cannot consider them completely satisfactory with respect to the more general intent of identifying the reasons why some of its complex extensions are so well-behaved. In particular, based only on the resulting model-theoretic and algorithmic features, we are not able to answer the question about the decidability of several multi-agent logics for strategic abilities, such as the *Alternating Temporal Logics* [2] ATL [58, 47] and ATL* [46] and the *one-goal fragment* SL[1G] [37] of *Strategy Logic* [12, 39, 38], which do not intrinsically embed such kinds of relativisation. For example, consider the ATL* formula $[\![\mathtt{a}, \mathtt{b}, \mathtt{c}]\!]\neg\psi$ over a game structure with $\mathtt{a}$, $\mathtt{b}$, $\mathtt{c}$, and $\mathtt{d}$ as the only agents. Intuitively, it asserts that agent $\mathtt{d}$ has a strategy, which depends upon those chosen by the other ones, ensuring that the LTL property ψ does not hold. Now, observe that the underlying strategic reasoning can be represented by the first-order sentence $\forall\mathtt{a}\forall\mathtt{b}\forall\mathtt{c}\exists\mathtt{d}\neg\mathtt{r}_\psi(\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d})$ having a prefix of the form $\forall^3\exists$ coupled with the quaternary atomic relation $\mathtt{r}_\psi$ in place of the temporal requirement ψ, which absorbs and hides the intrinsic second-order flavour of the original ATL* formula. It is evident that this sentence belongs neither to a decidable prefix-vocabulary class nor to the two-variable fragment. Moreover, quantifications are not guarded and negation is applied to a formula that is neither monadic nor guarded. Another explicit example is given by the SL[1G] sentence $[\![\mathtt{x}]\!]\langle\!\langle\mathtt{y}\rangle\!\rangle[\![\mathtt{z}]\!](\mathtt{a}, \mathtt{x})(\mathtt{b}, \mathtt{x})(\mathtt{c}, \mathtt{y})(\mathtt{d}, \mathtt{z})\psi$ asserting that, once $\mathtt{a}$ and $\mathtt{b}$ have chosen the common strategy $\mathtt{x}$, agent $\mathtt{c}$ can select its better response $\mathtt{y}$ to ensure ψ, in a way that is independent of the behaviour $\mathtt{z}$ of $\mathtt{d}$. In this case, the associated first-order sentence $\forall\mathtt{ab}\exists\mathtt{c}\forall\mathtt{d}\,\mathtt{r}_\psi(\mathtt{ab}, \mathtt{ab}, \mathtt{c}, \mathtt{d})$ has a prefix of the form $\forall\exists\forall$ coupled with the atomic relation $\mathtt{r}_\psi$, whose first two arguments are bound to the same variable. Again, we cannot cast this sentence in any of the decidable restrictions previously described. In particular, it is neither unary negation nor guarded negation, since universal quantifications are used as primary construct, which is not allowed in either of them.

Table 1 Notable algorithmic and model-theoretic properties for some fragments of FOL (M: *monadic*, 2VAR: 2-*variables*, GF: *guarded fragment*, CG: *clique guarded*, UN: *unary negation*, GN: *guarded negation*, FL: *fluted logic*, UF_1: *uniform one-dimensional fragment*, 1B: *one binding*, CB: *conjunctive binding*, DB: *disjunctive binding*, BB: *Boolean binding*).

	MC	SAT	FMP	CI	BD
FOL[M]	PSPACE-C	NEXPTIME-C [11]	✓ [11]	✓	✓
FOL[2VAR]	PTIME [56]	NEXPTIME-C [24]	✓ [40]	× [45]	× [45]
FOL[GF]	PTIME-C [9]	2EXPTIME-C [20]	✓ [20]	× [29]	✓ [29]
FOL[CG]	PSPACE-C [9]	2EXPTIME-C [19]	✓ [28]	× [29]	✓ [29]
FOL[UN]	$\Delta_2^{\text{P}}(\text{O}(\log^2 n))$-C [50]	2EXPTIME-C [50]	✓ [50]	✓ [50]	✓ [50]
FOL[GN]	$\Delta_2^{\text{P}}(\text{O}(\log^2 n))$-C [7]	2EXPTIME-C [7]	✓ [7]	✓ [4]	✓ [4]
FOL[FL]	PSPACE	NEXPTIME-C [42]	✓ [42]	✓ [42]	✓ [42]
FOL[UF_1]	?	NEXPTIME-C [34]	✓ [26]	?	?
FOL[1B]	PTIME [Thm.3.12]	Σ_3^{P}-C [Thm.3.13]	✓ [Thm.3.11]	✓ [Thm.3.11]	✓ [Thm.3.11]
FOL[CB]	PSPACE-C [Thm.3.8]	? [Con.3.10]	? [Con.3.10]	?	?
FOL[DB]	PSPACE-C [Thm.3.8]	Undecidable [Thm.3.9]	× [Thm.3.9]	?	?
FOL[BB]	PSPACE-C [Cor.3.6]	Undecidable [Cor.3.6]	× [Cor.3.6]	✓ [Cor.3.6]	✓ [Cor.3.6]

At this point, a question naturally arises: what are the *syntactic constraints* on the first-order representations of these logics of strategies that ensure their decidability? After a careful analysis, one can observe that such representations are always composed by a Boolean combination of sentences in prenex normal form, whose matrices are Boolean combinations of relations over the same arguments, which denote the agents of the game under analysis.

In this paper, trying to lay the foundation for a more thorough understanding of these decidability questions, we exploit the above observation to devise a new classification program based on the *binding forms* admitted in a sentence, *i.e.*, on the way the arguments of a relation can be bound to a variable. Indeed, inspired by the decoupling between agents and variables in SL, we define a syntactic variant of first-order logic in which arguments are bound to variables by means of an appropriate *binding construct*. To support this, similarly to the treatment of the attributes of a table in database theory [16], we describe a generalization of standard notions of *language signature* and *relational structure* in which arguments are explicit. With more detail, every relation r is associated with a set of arguments $\{a_1, \ldots, a_n\}$, which are bound to the variables via a binding form $(a_1, x_1) \cdots (a_n, x_n)r$ that replaces the standard writing $r(x_1, \ldots, x_n)$. So, for instance, a formula like $\mathtt{r}_1(\mathtt{x}_1, \mathtt{x}_2) \wedge \mathtt{r}_2(\mathtt{x}_2, \mathtt{x}_3) \rightarrow \mathtt{r}_3(\mathtt{x}_1, \mathtt{x}_3)$ would be written as $(\mathtt{a}_1, \mathtt{x}_1)(\mathtt{a}_2, \mathtt{x}_2)(\mathtt{a}_3, \mathtt{x}_3)(\mathtt{r}_1 \wedge \mathtt{r}_2 \rightarrow \mathtt{r}_3)$, assuming $\{\mathtt{a}_1, \mathtt{a}_2\}$, $\{\mathtt{a}_2, \mathtt{a}_3\}$, and $\{\mathtt{a}_1, \mathtt{a}_3\}$ as the arguments of $\mathtt{r}_1$, $\mathtt{r}_2$, and $\mathtt{r}_3$, respectively. Our notation, although perfectly equivalent to the classic one, allows to introduce and analyse, in a natural way, a hierarchy of four fragments of first-order logic based on the Boolean combinations of these forms. In particular, the simplest one, called *one binding*, is already incomparable with the clique guarded and guarded negation restrictions, as well as, with the *fluted logic* introduced by Quine [44] and the *uniform one-dimensional fragment* recently proposed by Hella, Kieronski, and Kuusisto [26, 34]. Examples of one-binding sentences result from the translation of the game properties described above, namely $\forall\mathtt{x}\forall\mathtt{y}\forall\mathtt{z}\exists\mathtt{w}(\mathtt{a}, \mathtt{x})(\mathtt{b}, \mathtt{y})(\mathtt{c}, \mathtt{z})(\mathtt{d}, \mathtt{w})\neg\mathtt{r}_\psi$ and $\forall\mathtt{x}\exists\mathtt{y}\forall\mathtt{z}(\mathtt{a}, \mathtt{x})(\mathtt{b}, \mathtt{x})(\mathtt{c}, \mathtt{y})(\mathtt{d}, \mathtt{z})\mathtt{r}_\psi$, where we assume that the relation $\mathtt{r}_\psi$ has the agents $\mathtt{a}$, $\mathtt{b}$, $\mathtt{c}$, and $\mathtt{d}$ as arguments. Via a novel model-theoretic technique exploiting the peculiarity of binding forms, we prove that our logic enjoys the *finite-model property*, both *Craig's interpolation* and *Beth's definability*, a PTIME *model checking* and a Σ_3^{P}-COMPLETE *satisfiability*.

In Table 1, we summarize results and open problems about classic and new fragments, from which we can immediately observe that the one-binding restriction is the simplest one with respect to the algorithmic point of view.

The main aim of this work is to describe a new criterion to classify formulas in order to better explore the boundary between *nice/non-nice* [3], *tame/untamed* [42], *easy/hard* [56], and *decidable/undecidable* [11] fragments of FOL, which is probably quite wider, but hopefully less irregular, than it was considered before. In particular, thanks to the introduced syntax, it is easier to discover connections with other languages, as those derived from *algebras* [13] and *calculi* [14] for relational databases. Moreover, we think it can help in the quest for the ultimate reasons behind tractability and decidability of a logic. Last but not least, we provide model-theoretic results that may be used as technical tools in other contexts, as well. Indeed, going back to the logics for strategic abilities, we claim that the *bounded-tree model property* of ATL [47], ATL* [46], and SL[1G] [37], shown to be crucial for their decidability, can be proved by means of the finite-model property of the one-binding fragment. This result is the focus of a dedicated work [36].

2 Signatures and Structures

Since from Codd's pioneering work on the definition of *relational databases* [13], several kinds of first-order languages have been used to describe databases queries [14]. In particular, *first-order logic* (FOL, for short) has been established as the main theoretical framework in which to prove results about properties of query languages [31, 55, 32]. In such a context, a table is usually represented as a mathematical relation between elements of a given domain, where its attributes are mapped to the indexes of that relation in a predetermined fixed way. Hence, attributes do not have any explicit matching element in the syntax of the language.

To introduce the *binding-form fragments* of FOL, we need to reformulate, instead, both the syntax and semantics of the logic in a way that is much closer to database theory. In particular, we explicitly associate a finite non-empty set of arguments to each relation [16], which are handled in the syntax via corresponding symbols. To do this, in the following, we introduce an alternative version of classic *language signatures* and *relational structures*.

Language Signatures. A *language signature* is a mathematical object describing the form of all non-logical symbols composing a formula. The typology we introduce here is purely relational, since we do not make use of constant or function symbols. Also, in our reasonings, we do not explicitly consider distinguished relations as equivalences, orders, or the equality.

▶ **Definition 2.1** (Language Signature). A *language signature* (LS, for short) is a tuple $\mathcal{L} \triangleq \langle \mathrm{Ar}, \mathrm{Rl}, \mathsf{ar} \rangle$, where Ar and Rl are the finite non-empty sets of *argument* and *relation* symbols and $\mathsf{ar} : \mathrm{Rl} \to 2^{\mathrm{Ar}} \setminus \{\emptyset\}$ is the *argument function* mapping every relation $r \in \mathrm{Rl}$ to its non-empty set of arguments $r^{\mathcal{L}} \triangleq \mathsf{ar}(r) \subseteq \mathrm{Ar}$.

Suppose we want to describe the schema of a genealogy database containing the relations *isFather*(*father*, *child*), *isMother*(*mother*, *child*), and *areParents*(*father*, *mother*, *child*). We can do this by means of the simple LS $\mathcal{L}_G = \langle \mathrm{Ar}, \mathrm{Rl}, \mathsf{ar} \rangle$, whose elements are set as follows: $\mathrm{Ar} = \{\mathtt{c} : \mathit{child}, \mathtt{f} : \mathit{father}, \mathtt{m} : \mathit{mother}\}$; $\mathrm{Rl} = \{\mathtt{Ft} : \mathit{isFather}, \mathtt{Mt} : \mathit{isMother}, \mathtt{Pr} : \mathit{areParents}\}$; $\mathtt{Ft}^{\mathcal{L}_G} = \{\mathtt{c}, \mathtt{f}\}$, $\mathtt{Mt}^{\mathcal{L}_G} = \{\mathtt{c}, \mathtt{m}\}$, and $\mathtt{Pr}^{\mathcal{L}_G} = \mathrm{Ar}$. From now on, we may put a superscript on a relation containing a list of its argument, *e.g.*, $\mathtt{Pr}^{\mathtt{cfm}}$, to help the reader to keep track of them.

Relational Structures. Given a language signature, we define the interpretation of all symbols by means of a *relational structure*, *i.e.*, a carrier domain together with an association

of each relation with a set of suitable tuples assuming values in that domain. Since relations with the same arity may have different arguments, it is not sufficient to work with elements of a Cartesian product as components of their interpretation. Therefore, we assign to every relation a set of *tuple functions* mapping arguments in their support to values of the carrier domain. Note that, under this definition, an order among arguments is not required.

▶ **Definition 2.2** (Relational Structure). A *relational structure* over an LS $\mathcal{L} = \langle \mathrm{Ar}, \mathrm{Rl}, \mathsf{ar} \rangle$ ($\mathcal{L}$-RS, for short) is a tuple $\mathcal{R} \triangleq \langle \mathrm{Dm}, \mathsf{rl} \rangle$, where Dm is the non-empty set of arbitrary objects named *domain* and $\mathsf{rl} : \mathrm{Rl} \to_r 2^{\mathsf{ar}(r) \to \mathrm{Dm}}$ is the *relation function* mapping every relation $r \in \mathrm{Rl}$ to the set $r^{\mathcal{R}} \triangleq \mathsf{rl}(r) \subseteq \mathsf{ar}(r) \to \mathrm{Dm}$ of *tuple functions* $\mathtt{t} \in \mathsf{rl}(r)$ from the arguments $a \in \mathsf{ar}(r)$ of r to values $\mathtt{t}(a) \in \mathrm{Dm}$ of the domain.

The *order* (*resp.*, *size*) of an $\mathcal{L}$-RS $\mathcal{R}$ is given by the cardinality $|\mathcal{R}| \triangleq |\mathrm{Dm}|$ (*resp.*, $\|\mathcal{R}\| \triangleq |\bigcup_{r \in \mathrm{Rl}} r^{\mathcal{R}}|$) of its domain set (*resp.*, relation function). A relational structure is *finite* if it has finite order and, so, a finite size.

Consider the LS $\mathcal{L}_G$ previously described. In Figure 1, we depict an $\mathcal{L}_G$-RS $\mathcal{R}_G$ containing part of Simpson Family's genealogy tree, where the two binary relations $\mathtt{Ft}^{\mathtt{cf}}$ and $\mathtt{Mt}^{\mathtt{cm}}$ are indicated through the edges labelled by their own names, while the ternary relation $\mathtt{Pr}^{\mathtt{cfm}}$ is described via the three hyperedges represented by the gray areas.

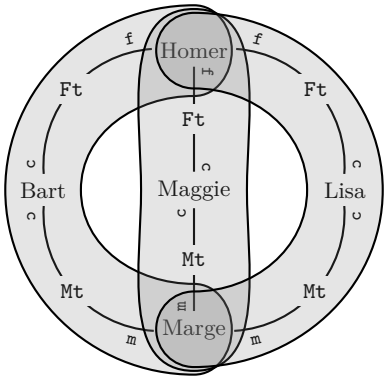


Figure 1 A relational structure.

3 First-Order Logic

We start by describing a slightly different but equivalent formalization of both syntax and semantics of FOL according to the explained alternatives of the language signature and relational structure. Then, we introduce a new family of fragments based on the kinds of binding forms allowed in a formula, *i.e.*, on the ways arguments can be bound to variables.

From now on, unless stated otherwise, we use $\mathcal{L} = \langle \mathrm{Ar}, \mathrm{Rl}, \mathsf{ar} \rangle$ to denote an *a priori* fixed LS. Also, Vr represents an enumerable non-empty set of *variables*. For the sake of succinctness, to indicate the extension of $\mathcal{L}$ with Vr, we adopt the composed symbol $\mathcal{L}(\mathrm{Vr})$.

Syntax. As far as the syntax of FOL is concerned, the novelty of our setting resides in the decoupling between variables and arguments, which implies an explicit occurrence of the latter as atomic components of a formula. Indeed, a variables x is not directly applied to the index associated with an argument a of a relation r, as in the usual writing $r(\ldots, x, \ldots)$, but an appropriate construct $(a, x)\varphi$, called *binding*, is required to bind a to x in the formula φ.

▶ **Definition 3.1** (FOL Syntax). FOL *formulas* over $\mathcal{L}(\mathrm{Vr})$ are built by means of the following context-free grammar, where $a \in \mathrm{Ar}$, $r \in \mathrm{Rl}$, and $x \in \mathrm{Vr}$:

$$\varphi := \bot \mid \top \mid r \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi) \mid \exists x.\varphi \mid \forall x.\varphi \mid (a, x)\varphi.$$

$\mathcal{L}(\mathrm{Vr})$-FOL denotes the set of all formulas over $\mathcal{L}(\mathrm{Vr})$ generated by the above grammar.

Consider again the LS $\mathcal{L}_G$ of Section 2 and suppose we want to formalize the fact that father and mother of a person are her parents and *vice versa*. This can be done via the $\mathcal{L}_G(\{\mathtt{x}, \mathtt{y}, \mathtt{z}\})$-FOL formula $\varphi_1 = \forall\mathtt{x}\forall\mathtt{y}\forall\mathtt{z}(\mathtt{f}, \mathtt{x})(\mathtt{m}, \mathtt{y})(\mathtt{c}, \mathtt{z})((\mathtt{Ft}^{\mathtt{cf}} \wedge \mathtt{Mt}^{\mathtt{cm}}) \leftrightarrow \mathtt{Pr}^{\mathtt{cfm}})$, where by $\varphi_1 \leftrightarrow \varphi_2$ we denote, as usual, the conjunction $(\varphi_1 \to \varphi_2) \wedge (\varphi_2 \to \varphi_1)$ with $\varphi_i \to \varphi_j$ in

place of $\neg\varphi_i \vee \varphi_j$. Now, imagine we need to determine which people have a father. To this aim, we can employ the formula $\varphi_2 = \exists \mathtt{x}(\mathtt{f},\mathtt{x})\mathtt{Ft}^{\mathtt{cf}}$, whose argument $\mathtt{c}$ is associated with the result. Finally, to query all pairs $(\mathtt{x},\mathtt{y})$ of grandfather and grandchild, we can use the formula $\varphi_3 = \exists \mathtt{z}((\mathtt{f},\mathtt{x})(\mathtt{c},\mathtt{z})\mathtt{Ft}^{\mathtt{cf}} \wedge (\mathtt{c},\mathtt{y})((\mathtt{f},\mathtt{z})\mathtt{Ft}^{\mathtt{cf}} \vee (\mathtt{m},\mathtt{z})\mathtt{Mt}^{\mathtt{cm}}))$. By fixing a linear order on the arguments, it is possible to rewrite every statement in the classic syntax. For instance, φ_1 can be expressed by $\forall \mathtt{x}\forall \mathtt{y}\forall \mathtt{z}((\mathtt{Ft}(\mathtt{x},\mathtt{z}) \wedge \mathtt{Mt}(\mathtt{y},\mathtt{z})) \leftrightarrow \mathtt{Pr}(\mathtt{x},\mathtt{y},\mathtt{z}))$, once it is assumed that $\mathtt{f} < \mathtt{m} < \mathtt{c}$. Conversely, every formula in the standard syntax can be translated into our syntax, by means of numeric arguments representing the positions in the relations. For example, the transitivity property $\forall \mathtt{x}\forall \mathtt{y}\forall \mathtt{z}((\mathtt{R}(\mathtt{x},\mathtt{y}) \wedge \mathtt{R}(\mathtt{y},\mathtt{z})) \rightarrow \mathtt{R}(\mathtt{x},\mathtt{z}))$ can be rewritten as $\forall \mathtt{x}\forall \mathtt{y}\forall \mathtt{z}(((1,\mathtt{x})(2,\mathtt{y})\mathtt{R}^{12} \wedge (1,\mathtt{y})(2,\mathtt{z})\mathtt{R}^{12}) \rightarrow (1,\mathtt{x})(2,\mathtt{z})\mathtt{R}^{12})$.

Usually, predicative logics, *i.e.*, languages having explicit quantifiers, need a concept of free or bound *placeholder* to formally evaluate the meaning of their formulas. The placeholders are used, in fact, to identify particular positions in a syntactic expression that are crucial for the definition of its semantics. Classic formalizations of FOL just require one kind of placeholder represented by the variables on which the formulas are built. In our new setting, instead, also the arguments have this fundamental role, as they are used to decouple variables from their association with a relation. As a consequence, we need a way to check whether a variable is quantified or an argument is bound. To do this, for every formula φ, we compute its set of *free arguments/variables* $\mathsf{free}(\varphi)$, as the subset of $\mathrm{Ar} \cup \mathrm{Vr}$ containing all arguments that are free from binding together with all variables occurring in some binding that are not quantified. Formally, we have the following definition.

▶ **Definition 3.2** (Free Placeholders). The set of *free arguments/variables* of an $\mathcal{L}(\mathrm{Vr})$-FOL formula can be computed via the function $\mathsf{free} : \mathcal{L}(\mathrm{Vr})\text{-FOL} \to 2^{\mathrm{Ar}\cup\mathrm{Vr}}$ defined as follows:

1. $\mathsf{free}(\bot) = \mathsf{free}(\top) \triangleq \emptyset$;
2. $\mathsf{free}(r) \triangleq \mathsf{ar}(r)$, where $r \in \mathrm{Rl}$;
3. $\mathsf{free}(\neg\varphi) \triangleq \mathsf{free}(\varphi)$;
4. $\mathsf{free}(\varphi_1 \mathtt{Op} \varphi_2) \triangleq \mathsf{free}(\varphi_1) \cup \mathsf{free}(\varphi_2)$, where $\mathtt{Op} \in \{\wedge, \vee\}$;
5. $\mathsf{free}(\mathtt{Qn}x.\varphi) \triangleq \mathsf{free}(\varphi) \setminus \{x\}$, where $\mathtt{Qn} \in \{\exists, \forall\}$;
6. $\mathsf{free}((a,x)\varphi) \triangleq \begin{cases} (\mathsf{free}(\varphi) \setminus \{a\}) \cup \{x\}, & \text{if } a \in \mathsf{free}(\varphi); \\ \mathsf{free}(\varphi), & \text{otherwise.} \end{cases}$

A formula φ without free arguments (*resp.*, variables), *i.e.*, $\mathsf{ar}(\varphi) \triangleq \mathsf{free}(\varphi) \cap \mathrm{Ar} = \emptyset$ (*resp.*, $\mathsf{vr}(\varphi) \triangleq \mathsf{free}(\varphi) \cap \mathrm{Vr} = \emptyset$), is named *argument* (*resp.*, *variable*) *closed*. If φ is both argument and variable closed, it is referred to as a *sentence*. Consider the three formulas φ_1, φ_2, and φ_3 given above. We have that $\mathsf{free}(\varphi_1) = \emptyset$, $\mathsf{free}(\varphi_2) = \mathsf{ar}(\varphi_2) = \{\mathtt{c}\}$, and $\mathsf{free}(\varphi_3) = \mathsf{vr}(\varphi_3) = \{\mathtt{x},\mathtt{y}\}$. Thus, φ_1 is a sentence, φ_2 is variable closed, and φ_3 is argument closed.

One may observe that the proposed syntax has some similarities with the relational calculus introduced by Codd [14, 16] as a logic counterpart of standard relational algebra [13], since in this language the attributes may be identified by name rather than position. However, its notation is tuple-centric, thus, it necessarily requires the use of the equality relation even to express very simple properties that do not intrinsically need it. For example, to describe in that calculus the left-totality of a binary relation $\mathtt{r}$ over the arguments $\mathtt{a}$ and $\mathtt{b}$, we have to write $\forall \mathtt{t}\exists \mathtt{t}'(\mathtt{t}[\mathtt{b}] = \mathtt{t}'[\mathtt{a}] \wedge \mathtt{r}(\mathtt{t}'))$ (unrestricted semantics) or $\forall \mathtt{t}(\mathtt{r}(\mathtt{t}) \rightarrow \exists \mathtt{t}'(\mathtt{t}[\mathtt{b}] = \mathtt{t}'[\mathtt{a}] \wedge \mathtt{r}(\mathtt{t}')))$ (active-domain semantics). In our syntax, instead, we just write $\forall \mathtt{x}\exists \mathtt{y}(\mathtt{a},\mathtt{x})(\mathtt{b},\mathtt{y})\mathtt{r}$ (unrestricted semantics) or $\forall \mathtt{z}\forall \mathtt{x}((\mathtt{a},\mathtt{z})(\mathtt{b},\mathtt{x})\mathtt{r} \rightarrow \exists \mathtt{y}(\mathtt{a},\mathtt{x})(\mathtt{b},\mathtt{y})\mathtt{r})$ (active-domain semantics).

Semantics. The semantics of FOL described here is defined, as usual, *w.r.t.* an RS. As a matter of fact, the peculiarities of our setting only concern the interpretation of binding constructs and the non-standard evaluation of relations.

In order to formalize the meaning of a formula, we first need to describe the concept of *assignment*, *i.e.*, a partial function $\chi \in \mathrm{Asg}_{\mathrm{D}} \triangleq (\mathrm{Ar} \cup \mathrm{Vr}) \rightharpoonup \mathrm{D}$ mapping each placeholder in its domain to a value of an arbitrary set D, which is used to define a valuation of all the free arguments and variables. For a given placeholder $p \in \mathrm{Ar} \cup \mathrm{Vr}$ and a value $d \in \mathrm{D}$, the notation $\chi[p \mapsto d]$ represents the assignment defined on $\mathsf{dom}(\chi[p \mapsto d]) \triangleq \mathsf{dom}(\chi) \cup \{p\}$ that returns d on p and is equal to χ on the remaining part of its domain, *i.e.*, $\chi[p \mapsto d](p) \triangleq d$ and $\chi[p \mapsto d](p') \triangleq \chi(p')$, for all $p' \in \mathsf{dom}(\chi) \setminus \{p\}$.

▶ **Definition 3.3** (FOL Semantics)**.** Let $\mathcal{R}$ be an $\mathcal{L}$-RS and φ an $\mathcal{L}(\mathrm{Vr})$-FOL formula. Then, for all assignments $\chi \in \mathrm{Asg}_{\mathrm{Dm}}$ with $\mathsf{free}(\varphi) \subseteq \mathsf{dom}(\chi)$, the relation $\mathcal{R}, \chi \models \varphi$ is inductively defined on the structure of φ as follows.

1. Boolean values and connectives are interpreted as usual.
2. $\mathcal{R}, \chi \models r$ if $\chi_{\restriction r^{\mathcal{L}}} \in r^{\mathcal{R}}$, for every relation $r \in \mathrm{Rl}$.
3. For each variable $x \in \mathrm{Vr}$, it is set that:
   a. $\mathcal{R}, \chi \models \exists x.\varphi$ if there exists a value $d \in \mathrm{Dm}$ such that $\mathcal{R}, \chi[x \mapsto d] \models \varphi$;
   b. $\mathcal{R}, \chi \models \forall x.\varphi$ if, for all values $d \in \mathrm{Dm}$, it holds that $\mathcal{R}, \chi[x \mapsto d] \models \varphi$.
4. $\mathcal{R}, \chi \models (a, x)\varphi$ if $\mathcal{R}, \chi[a \mapsto \chi(x)] \models \varphi$, for each argument $a \in \mathrm{Ar}$ and variable $x \in \mathrm{Vr}$.

Intuitively, Condition 2 states that a relation r is satisfied by an assignment χ whenever the tuple function $\chi_{\restriction r^{\mathcal{L}}}$ obtained by the restriction of χ to the arguments $r^{\mathcal{L}}$ of r is included in the interpretation $r^{\mathcal{R}}$. Condition 4, instead, interprets the binding construct (a, x) by associating the argument a with the value of the variable x contained inside the assignment.

Consider again the formulas φ_1, φ_2, and φ_3 and the $\mathcal{L}_G$-RS $\mathcal{R}_G$ of Figure 1. We have that $\mathcal{R}_G, \varnothing \models \varphi_1$. Moreover, $\mathcal{R}_G, \varnothing[\mathtt{c} \mapsto \mathrm{Lisa}] \models \varphi_2$ and $\mathcal{R}_G, \varnothing[\mathtt{x} \mapsto \mathrm{Homer}, \mathtt{y} \mapsto \mathrm{Bart}] \not\models \varphi_3$.

To complete the description of the semantics, we give the notions of *model* and *satisfiability*. For an $\mathcal{L}$-RS $\mathcal{R}$ and an $\mathcal{L}(\mathrm{Vr})$-FOL sentence φ, we say that $\mathcal{R}$ is a *model* of φ, in symbols $\mathcal{R} \models \varphi$, iff $\mathcal{R}, \varnothing \models \varphi$, where $\varnothing \in \mathrm{Asg}_{\mathrm{Dm}}$ simply denotes the empty assignment. We also say that φ is *satisfiable* iff there exists a model for it. Given two $\mathcal{L}(\mathrm{Vr})$-FOL formulas φ_1 and φ_2, we say that φ_1 *implies* φ_2, in symbols $\varphi_1 \Rightarrow \varphi_2$, iff $\mathcal{R}, \chi \models \varphi_1$ implies $\mathcal{R}, \chi \models \varphi_2$, for each $\mathcal{L}$-RS $\mathcal{R}$ and assignment $\chi \in \mathrm{Asg}_{\mathrm{Dm}}$ with $\mathsf{free}(\varphi_1), \mathsf{free}(\varphi_2) \subseteq \mathsf{dom}(\chi)$. Moreover, we say that φ_1 is *equivalent* to φ_2, in symbols $\varphi_1 \equiv \varphi_2$, iff both $\varphi_1 \Rightarrow \varphi_2$ and $\varphi_2 \Rightarrow \varphi_1$ hold.

Fragments. We now introduce a family of syntactic fragments of FOL by means of a special *normal form*, where relations over the same set of arguments may be clustered together by a unique sequence of bindings. With more detail, we consider Boolean combinations of sentences in prenex normal form in which quantification prefixes are coupled with Boolean combinations of these relation clusters called *binding forms*. Each fragment is then characterized by a specific constraint on the possible combinations of these forms.

A *quantification prefix* $\wp \in \mathrm{Qn} \subseteq \{\exists x, \forall x : x \in \mathrm{Vr}\}^*$ is a finite sequence of quantifiers, in which each variable occurs at most once. Similarly, a *binding prefix* $\flat \in \mathrm{Bn} \subseteq \{(a, x) : a \in \mathrm{Ar} \wedge x \in \mathrm{Vr}\}^*$ is a finite sequence of bindings, in which each argument occurs at most once. For example, $\wp = \forall\mathtt{x}\forall\mathtt{y}\forall\mathtt{z}$ and $\flat = (\mathtt{f}, \mathtt{x})(\mathtt{m}, \mathtt{y})(\mathtt{c}, \mathtt{z})$ are the quantification and binding prefixes occurring in the formula $\varphi_1 = \wp\flat((\mathtt{Ft}^{\mathtt{cf}} \wedge \mathtt{Mt}^{\mathtt{cm}}) \leftrightarrow \mathtt{Pr}^{\mathtt{cfm}})$ previously described. Finally, a *derived relation* $\widehat{r} \in \widehat{\mathrm{Rl}}$ is a Boolean combination of relations in Rl all having the same arguments, while a *binding form* $\flat\widehat{r} \in \mathrm{BF}$ is an argument-closed formula obtained by the juxtaposition of a binding prefix to a derived relation. Note that $\flat((\mathtt{Ft}^{\mathtt{cf}} \wedge \mathtt{Mt}^{\mathtt{cm}}) \leftrightarrow \mathtt{Pr}^{\mathtt{cfm}})$ is not a binding form, as the three relations have different arguments.

▶ **Definition 3.4** (Binding-Form Fragments)**.** *Boolean-binding formulas* over $\mathcal{L}(\mathrm{Vr})$ are built by means of the following context-free grammar, where $\wp \in \mathrm{Qn}$ and $\flat\widehat{r} \in \mathrm{BF}$:

$$\varphi := \bot \mid \top \mid \wp\psi \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi); \qquad \psi := \flat\widehat{r} \mid (\psi \wedge \psi) \mid (\psi \vee \psi).$$

$\mathcal{L}$(Vr)-FOL[BB] denotes the enumerable set of all formulas over $\mathcal{L}$(Vr) generated by the principal rule φ. Moreover, the *conjunctive*, *disjunctive*, and *one binding* fragments of FOL (FOL[CB], FOL[DB], and FOL[1B], for short) are obtained, respectively, by weakening the secondary rule ψ as follows: $\psi := \flat\widehat{r} \mid (\psi \wedge \psi)$, $\psi := \flat\widehat{r} \mid (\psi \vee \psi)$, and $\psi := \flat\widehat{r}$.

As an example, consider the FOL[BB] sentence $\forall \mathtt{x}\forall \mathtt{y}\forall \mathtt{z}(((\mathtt{c},\mathtt{x})(\mathtt{f},\mathtt{y})\mathtt{Ft}^{\mathtt{cf}} \wedge (\mathtt{c},\mathtt{x})(\mathtt{m},\mathtt{z})\mathtt{Mt}^{\mathtt{cm}}) \leftrightarrow (\mathtt{c},\mathtt{x})(\mathtt{f},\mathtt{y})(\mathtt{m},\mathtt{z})\mathtt{Pr}^{\mathtt{cfm}})$. It is easy to see that this is equivalent to the formula φ_1 given above. In general, by applying a simple generalization of the classic procedure used to obtain a prenex normal form, which further pushes the bindings inside as much as possible, we can always transform a FOL formula into an equivalent FOL[BB] one, with only a linear blow-up.

▶ **Theorem 3.5** (Binding Normal Form). *For each* $\mathcal{L}$(Vr)-FOL *formula, there exists an equivalent* $\mathcal{L}$(Vr)-FOL[BB] *one.*

An immediate consequence of the previous theorem is that FOL[BB] inherits all computational and model-theoretic properties of FOL. Therefore, the following holds.

▶ **Corollary 3.6** (FOL[BB] Properties). FOL[BB] *does enjoy both Craig's interpolation and Beth's definability, but not the finite-model property. Also, it has a* PSPACE-COMPLETE *model-checking problem and an undecidable satisfiability problem.*

At this point, we describe some meaningful example to illustrate the expressive power of the other binding-form fragments. First extend the LS $\mathcal{L}_G$ of Section 2, by adding the two arguments $\mathtt{p}_1$ and $\mathtt{p}_2$, standing for "*person*", and the three binary relations $\mathtt{Sb}$, $\mathtt{Mr}$, and $\mathtt{Lv}$ with $\mathtt{Sb}^{\mathcal{L}_G} = \mathtt{Mr}^{\mathcal{L}_G} = \mathtt{Lv}^{\mathcal{L}_G} = \{\mathtt{p}_1, \mathtt{p}_2\}$, in place of "*Sibling*", "*Married*", and "*inLove*", respectively. By definition, two siblings share the same parents. This can be expressed by the FOL[DB] sentence $\forall \mathtt{x}\forall \mathtt{y}\forall \mathtt{z}\forall \mathtt{w}(((\mathtt{p}_1,\mathtt{x})(\mathtt{p}_2,\mathtt{y})\mathtt{Sb}^{\mathtt{p}_1\mathtt{p}_2} \wedge (\mathtt{c},\mathtt{x})(\mathtt{f},\mathtt{z})(\mathtt{m},\mathtt{w})\mathtt{Pr}^{\mathtt{cfm}}) \rightarrow (\mathtt{c},\mathtt{y})(\mathtt{f},\mathtt{z})(\mathtt{m},\mathtt{w})\mathtt{Pr}^{\mathtt{cfm}})$, where, for the sake of readability, we use the form $(\psi_1 \wedge \psi_2) \rightarrow \psi_3$ to represent $\neg\psi_1 \vee \neg\psi_2 \vee \psi_3$. In the romantic literature it is common to find an unrequited love, whose scenario may be represented with the FOL[CB] sentence $\exists \mathtt{x}\exists \mathtt{y}\exists \mathtt{z}((\mathtt{p}_1,\mathtt{x})(\mathtt{p}_2,\mathtt{y})\mathtt{Lv}^{\mathtt{p}_1\mathtt{p}_2} \wedge (\mathtt{p}_1,\mathtt{y})(\mathtt{p}_2,\mathtt{x})\neg\mathtt{Lv}^{\mathtt{p}_1\mathtt{p}_2} \wedge (\mathtt{p}_1,\mathtt{y})(\mathtt{p}_2,\mathtt{z})\mathtt{Lv}^{\mathtt{p}_1\mathtt{p}_2})$. Usually, two married people cannot be sibling and should be in love. The FOL[1B] sentence $\forall \mathtt{x}\forall \mathtt{y}(\mathtt{p}_1,\mathtt{x})(\mathtt{p}_2,\mathtt{y})(\mathtt{Mr}^{\mathtt{p}_1\mathtt{p}_2} \rightarrow \neg\mathtt{Sb}^{\mathtt{p}_1\mathtt{p}_2} \wedge \mathtt{Lv}^{\mathtt{p}_1\mathtt{p}_2})$ precisely expresses this fact. To conclude, consider the FOL[DB] sentence $\forall \mathtt{x}\forall \mathtt{y}((\mathtt{p}_1,\mathtt{x})(\mathtt{p}_2,\mathtt{y})\mathtt{Mr}^{\mathtt{p}_1\mathtt{p}_2} \rightarrow (\mathtt{p}_1,\mathtt{y})(\mathtt{p}_2,\mathtt{x})\mathtt{Mr}^{\mathtt{p}_1\mathtt{p}_2})$ stating that the relation $\mathtt{Mr}$ is symmetric. The only reason why we cannot express it in FOL[1B] is that the two bindings are permutations of each other. Therefore, if we allow the use of permutations in the definition of derived relations, we obtain a strictly more expressive FOL[1B] fragment able to describe the symmetric property as follows: $\forall \mathtt{x}\forall \mathtt{y}(\mathtt{p}_1,\mathtt{x})(\mathtt{p}_2,\mathtt{y})(\mathtt{Mr}^{\mathtt{p}_1\mathtt{p}_2} \rightarrow \{\mathtt{p}_1\mathtt{p}_2 \mapsto \mathtt{p}_2\mathtt{p}_1\}\mathtt{Mr}^{\mathtt{p}_1\mathtt{p}_2})$. By using techniques similar to those developed for FOL[1B], one can prove that such an extension retains exactly the same model-theoretic and algorithmic properties. However, for the sake of simplicity, we prefer to postpone this study to the extended version of the paper.

Before continuing, we want to point out an interesting connection between FOL[1B] and the language par excellence for relational databases. Indeed, without considering negation, it is not hard to see that sentences of our logic correspond to expressions of relational algebra of the form unions/natural joins of projections/divisions of positive Boolean combinations of atomic relations. The connection is still valid in the presence of negation, but it requires a deeper analysis, as observed in Section 4. Unfortunately, this correspondence is not helpful in the derivation of the model-theoretic and satisfiability results described in the following.

An expert reader might also have noticed a significant similarity between the concept of binding form and the notion of uniformity formalized in [26], which is used to introduce

the uniform one-dimensional fragment of FOL. Nevertheless, the syntax of the four binding fragments does not comply with the further required one-dimensional restriction. Therefore, these logics should be orthogonal *w.r.t.* the expressive power. However, a further analysis of connections and differences is in order.

Results. The negative features concerning FOL[BB] have spurred us to investigate the three simpler fragments FOL[CB], FOL[DB], and FOL[1B], to which we devote the remaining part of this section by giving an overview of the obtained model-theoretic and algorithmic results.

As far as the expressiveness is concerned, we show that FOL[1B] is strictly less expressive than FOL[CB] and FOL[DB]. This is done by means of a suitable concept of bisimulation under which the first fragment is proved to be invariant. Also, FOL[1B] is incomparable with other logics studied in the literature. Indeed, on the one hand, by means of the sentence $\forall \mathtt{x} \forall \mathtt{y} \forall \mathtt{z} (\mathtt{a}_1, \mathtt{x})(\mathtt{a}_2, \mathtt{y})(\mathtt{a}_3, \mathtt{z}) \mathtt{r}^{\mathtt{a}_1 \mathtt{a}_2 \mathtt{a}_3}$, we can state the completeness of a ternary relation $\mathtt{r}$, which is not possible to express in any of the fragments FOL[2VAR], FOL[CG], and FOL[GN]. Moreover, FOL[FL] cannot express the reflexivity of a binary relation $\mathtt{r}$ [43], which is easily described by the FOL[1B] sentence $\forall \mathtt{x} (\mathtt{a}_1, \mathtt{x})(\mathtt{a}_2, \mathtt{x}) \mathtt{r}^{\mathtt{a}_1 \mathtt{a}_2}$. On the other hand, due to the invariance under the particular bisimulation referred to above, we have shown that the simple modal logic formula $\Box \Diamond \mathtt{p}$, does not have an equivalent formulation in FOL[1B]. Finally, it is possible to observe that both FOL[CB] and FOL[DB] are not closed under negation and the former, differently from the latter, strictly subsumes the conjunctive query fragment of FOL.

▶ **Theorem 3.7** (Expressiveness). FOL[1B] *is strictly less expressive than* FOL[CB] *and* FOL[DB] *and incomparable with* FOL[2VAR], FOL[CG], FOL[GN], FOL[FL].

Although FOL[CB] and FOL[DB] have a more constrained syntax than FOL[BB], they do not have an easier model-checking problem. We can prove this by employing the classic reduction from the satisfiability problem of QBF, which is known to be PSPACE-COMPLETE.

▶ **Theorem 3.8** (FOL[CB] & FOL[DB] MC). *Both* FOL[CB] *and* FOL[DB] *have a* PSPACE-COMPLETE *model-checking problem.*

For FOL[DB], the situation is even worse. In fact, it is not hard to see that this fragment does not enjoy the finite-model property, as we can express the existence of an unbounded strict partial order [15]. Moreover, we can show that it is a conservative reduction class [11].

▶ **Theorem 3.9** (FOL[DB] FMP & SAT). FOL[DB] *does not enjoy the finite-model property and has an undecidable satisfiability problem.*

Once observed that the negation of a FOL[CB] formula is equivalent to a FOL[DB] one and *vice versa*, we immediately derive that the validity problem for FOL[CB] is undecidable. Nevertheless, we conjecture that this logic is model-theoretically and algorithmically well-behaved, as we think that the techniques developed for FOL[1B] can be suitably adapted to work with the more expressive fragment as well.

▶ **Conjecture 3.10** (FOL[CB] FMP & SAT). FOL[CB] *does enjoy the finite-model property and has a decidable satisfiability problem.*

We now focus on FOL[1B]. First of all, we prove that it is model-theoretically well-behaved, as it enjoys the finite-model property and both Craig's interpolation and Beth's definability, which are considered as mandatory properties for a "nice" FOL fragment [3]. To do this, we devise a novel technique that allows us to determine which subsentences of a given sentence might prevent its satisfiability. In other words, we propose a criterion that

identifies a set of *templates* of a formula, *i.e.*, pairs of quantification and binding prefixes of its subformulas, that may enforce inconsistent requirements on part of the underlying model. Such a set of templates is said to be *overlapping*. For example, consider the sentence $\forall \mathtt{x} \forall \mathtt{y} (\mathtt{a}_1, \mathtt{x})(\mathtt{a}_2, \mathtt{y}) \mathtt{r}^{\mathtt{a}_1 \mathtt{a}_2} \wedge \exists \mathtt{x} (\mathtt{a}_1, \mathtt{x})(\mathtt{a}_2, \mathtt{x}) \neg \mathtt{r}^{\mathtt{a}_1 \mathtt{a}_2}$. It is immediate to see that it is not satisfiable, as there is an assignment of the arguments $\mathtt{a}_1$ and $\mathtt{a}_2$, shared by both the templates $(\forall \mathtt{x} \forall \mathtt{y}, (\mathtt{a}_1, \mathtt{x})(\mathtt{a}_2, \mathtt{y}))$ and $(\exists \mathtt{x}, (\mathtt{a}_1, \mathtt{x})(\mathtt{a}_2, \mathtt{x}))$, which leads to the inconsistent formula $\mathtt{r}^{\mathtt{a}_1 \mathtt{a}_2} \wedge \neg \mathtt{r}^{\mathtt{a}_1 \mathtt{a}_2}$. These templates are, in fact, overlapping. By exploiting this criterion, we are able to build a particular kind of a finite Herbrand structure [8, 54] that satisfies the formula iff the overlapping templates only require consistent properties on the shared assignments. This is the basis for the finite-model property. The same tool is also used to find a particular normal form for FOL[1B] that allows us to prove Craig's interpolation, from which Beth's definability immediately follows. With more detail, we reduce the search for an interpolant between two FOL[1B] sentences φ_1 and φ_2 to the same problem between two propositional formulas η_1 and η_2 suitably obtained from the derived relations composing the matrices of the former. As for the standard syntax, in the new one, an interpolant is a sentence φ over the signature common to φ_1 and φ_2 such that $\varphi_1 \Rightarrow \varphi \Rightarrow \varphi_2$ holds. For instance, consider the implication $\exists \mathtt{x} \forall \mathtt{y} \flat (\mathtt{p} \wedge (\mathtt{q} \rightarrow \mathtt{r})) \wedge \forall \mathtt{x} \exists \mathtt{y} \flat (\mathtt{p} \wedge (\mathtt{r} \rightarrow \mathtt{q})) \Rightarrow \exists \mathtt{x} \exists \mathtt{y} \flat (\mathtt{s} \rightarrow (\mathtt{q} \leftrightarrow \mathtt{r}))$, where $\flat = (\mathtt{x}, \mathtt{a}_1)(\mathtt{y}, \mathtt{a}_2)$. Now, it is not hard to see that $\exists \mathtt{x} \exists \mathtt{y} \flat (\mathtt{q} \leftrightarrow \mathtt{r})$ is the associated FOL[1B] interpolant, where $\mathtt{q} \leftrightarrow \mathtt{r}$ is obtained as the propositional one for $(\mathtt{p} \wedge (\mathtt{q} \rightarrow \mathtt{r})) \wedge (\mathtt{p} \wedge (\mathtt{r} \rightarrow \mathtt{q})) \Rightarrow (\mathtt{s} \rightarrow (\mathtt{q} \leftrightarrow \mathtt{r}))$.

▶ **Theorem 3.11** (FOL[1B] FMP, CI & BD). FOL[1B] *does enjoy the finite-model property and both Craig's interpolation and Beth's definability.*

FOL[1B] is very well-behaved from the algorithmic point of view too. Indeed, we can provide a PTIME model-checking procedure *w.r.t.* the combined complexity, which is based on a linear reduction to a two-player reachability game, whose solution is known to be in PTIME [30]. It remains open whether the problem is also hard for this class.

▶ **Theorem 3.12** (FOL[1B] MC). FOL[1B] *has a* PTIME *model-checking problem.*

We finally discuss the satisfiability problem for FOL[1B], which we prove to be complete for the third level of the polynomial hierarchy, *i.e.*, $\Sigma^{\mathrm{P}}_3 = \mathrm{NPTIME}^{\mathrm{CONPTIME}^{\mathrm{NPTIME}}}$. In other words, it is solvable by an NPTIME Turing machine having access to a CONPTIME oracle that, in turn, can exploit an NPTIME advice. For the upper bound, thanks to the characterization of satisfiability via the overlapping templates mentioned above, we are able to reduce the problem to a three-round two-player game, in which each player has either NPTIME or CONPTIME computational power. The lower bound follows from a reduction from the satisfiability problem of QBF sentences with alternation 2. Intuitively, we transform a formula of the form $\exists^* \forall^* \exists^* \psi$ into an equisatisfiable conjunction $\varphi_\exists \wedge \varphi_\forall \wedge \varphi_\psi$ of FOL[1B] sentences such that the first two take care of the initial existential and universal quantifications, while the last handles the matrix. Observe that, since FOL[1B] is closed under negation, its *validity* and *implication problems* inherit the same complexity.

▶ **Theorem 3.13** (FOL[1B] SAT). FOL[1B] *has a* Σ^{P}_3-COMPLETE *satisfiability problem.*

For the sake of space, we exclusively devote the remaining part of this work to sketching the proofs of the two algorithmic results concerning FOL[1B].

4 Model Checking

We now describe a PTIME model-checking procedure for FOL[1B] sentences of unbounded width based on a reduction to a reachability game [30] over a tree arena, whose depth and

size are bound by, respectively, the number of quantifiers in the sentence and the size of the underlying RS. It is worth noting that the verification of a sentence φ can be reduced to a linear number of checks of its subsentences of the form $\wp\flat\widehat{r}$. Thus, we just focus on the latter.

Matrix Normalization. Before starting, we need to introduce the concept of restricted Boolean combination, *i.e.*, a Boolean formula in which the negation is replaced by the difference connective $\setminus$, whose semantics is set as follows: $\phi_1 \setminus \phi_2 \triangleq \phi_1 \wedge \neg\phi_2$. Now, we can show that, for each derived relation $\widehat{r}$, there exists a restricted Boolean combination $\widehat{r}^\star$ with $|\widehat{r}^\star| = \mathrm{O}(|\widehat{r}|)$ such that either $\widehat{r} \equiv \widehat{r}^\star$ or $\widehat{r} \equiv \neg\widehat{r}^\star$. This can be proved by induction via De Morgan's laws and the substitutions of $\varphi_1 \wedge \neg\varphi_2$ and $\varphi_1 \vee \neg\varphi_2$ with $\varphi_1 \setminus \varphi_2$ and $\neg(\varphi_2 \setminus \varphi_1)$, respectively. In case $\widehat{r} \equiv \widehat{r}^\star$, it holds that $\wp\flat\widehat{r} \equiv \wp\flat\widehat{r}^\star$. If $\widehat{r} \equiv \neg\widehat{r}^\star$, instead, we have that $\wp\flat\widehat{r} \equiv \neg\overline{\wp}\flat\widehat{r}^\star$, where the quantification prefix $\overline{\wp}$ is the dual of $\wp$, *i.e.*, every existential (*resp.*, universal) quantifier in $\wp$ is replaced by a universal (*resp.*, existential) one in $\overline{\wp}$. Hence, *w.l.o.g.*, we can assume $\widehat{r}$ to be a restricted Boolean combination of relations.

Model-Checking Procedure. The first step in our procedure is to transform the restricted derived relation $\widehat{r}$ into a fresh atomic relation $r^\star$ containing all and only the assignments satisfying $\widehat{r}$. From the original LS $\mathcal{L}{=}\langle \mathrm{Ar}, \mathrm{Rl}, \mathsf{ar}\rangle$ and $\mathcal{L}$-RS $\mathcal{R}{=}\langle \mathrm{Dm}, \mathsf{rl}\rangle$, we build the new LS $\mathcal{L}^\star \triangleq \langle \mathsf{ar}(\widehat{r}), \{r^\star\}, \mathsf{ar}^\star\rangle$ with $\mathsf{ar}^\star(r^\star) \triangleq \mathsf{ar}(\widehat{r})$ and the $\mathcal{L}^\star$-RS $\mathcal{R}^\star \triangleq \langle \mathrm{Dm}, \mathsf{rl}^\star\rangle$ with $\mathsf{rl}^\star(r^\star) \triangleq \mathsf{v}(\widehat{r})$, where the valuation function v is set as follows: *(i)* $\mathsf{v}(r) \triangleq \mathsf{rl}(r)$; *(ii)* $\mathsf{v}(\phi_1 \wedge \phi_2) \triangleq \mathsf{v}(\phi_1) \cap \mathsf{v}(\phi_2)$; *(iii)* $\mathsf{v}(\phi_1 \vee \phi_2) \triangleq \mathsf{v}(\phi_1) \cup \mathsf{v}(\phi_2)$; *(iv)* $\mathsf{v}(\phi_1 \setminus \phi_2) \triangleq \mathsf{v}(\phi_1) \setminus \mathsf{v}(\phi_2)$. It is easy to observe that $\mathcal{R} \models \wp\flat\widehat{r}$ iff $\mathcal{R}^\star \models \wp\flat r^\star$. Moreover, such a construction can be done in time $\mathrm{O}(|\widehat{r}| \cdot \|\mathcal{R}\|)$. Note that, in this step, we are just employing the classic set operations of relational algebra.

The second step is devoted to the transformation of the binding form $\flat r^\star$ into an injective one $\flat^\natural r^\natural$ having $\flat^\natural \triangleq \prod_{x \in \mathsf{free}(\flat r^\star)}(x, x)$, *i.e.*, a formula in which every argument is bound to a different variable. From the previous LS $\mathcal{L}^\star$ and $\mathcal{L}^\star$-RS $\mathcal{R}^\star$, we build the new LS $\mathcal{L}^\natural \triangleq \langle \mathsf{free}(\flat r^\star), \{r^\natural\}, \mathsf{ar}^\natural\rangle$, where $\mathsf{ar}^\natural(r^\natural) \triangleq \mathsf{free}(\flat r^\star)$, and the $\mathcal{L}^\natural$-RS $\mathcal{R}^\natural \triangleq \langle \mathrm{Dm}, \mathsf{rl}^\natural\rangle$ with $\mathsf{rl}^\natural(r^\natural) \triangleq \{\chi \in \mathrm{Asg}_{\mathrm{Dm}} : \mathcal{R}^\star, \chi \models \flat r^\star\}$. Now, we immediately obtain that $\mathcal{R}^\star \models \wp\flat r^\star$ iff $\mathcal{R}^\natural \models \wp\flat^\natural r^\natural$. Moreover, the construction can be done in time $\mathrm{O}(|\flat| \cdot \|\mathcal{R}\|)$. In this case, we are making use of the three relational algebra operations of selection, projection, and renaming.

With the third and final step, we reduce the verification of $\mathcal{R}^\natural \models \wp\flat^\natural r^\natural$ to the win of the first player in a suitably constructed reachability game. Firstly, fix an ordering $<\subseteq \mathrm{Dm}\times\mathrm{Dm}$ among the values of the carrier domain Dm. Then, let $<_\wp$ be the ordering on variables induced by the quantification prefix $\wp$. Now, we can sort in the lexicographic order $(<, <_\wp)$ all tuple functions contained into the interpretation $\mathsf{rl}^\natural(r^\natural)$ of the relation $r^\natural$. At this point, from these tuples, we can construct a prefix-tree data structure, *a.k.a.* trie [17], taking values in the set $\{\bot, \top\}$. In particular, to each tuple in $\mathsf{rl}^\natural(r^\natural)$ we assign $\top$, while to every minimal partial tuple that does not have an extension in the interpretation we assign $\bot$. As an example, in Figure 2, we depict the trie corresponding to $\mathrm{Dm} = \{1, 2, 3\}$, $\mathsf{ar}^\natural(r^\natural) = \{\mathtt{x}, \mathtt{y}, \mathtt{z}\}$, $\mathtt{x}{<_\wp}\mathtt{y}{<_\wp}\mathtt{z}$, and $\mathsf{rl}^\natural(r^\natural) = \{\mathtt{xyz} \mapsto 111, \mathtt{xyz} \mapsto 123, \mathtt{xyz} \mapsto 132, \mathtt{xyz} \mapsto 333\}$. Note that, as there are no tuple functions in $\mathsf{rl}^\natural(r^\natural)$ having the variable $\mathtt{x}$ set to 2, it follows that the corresponding node of the tree has a $\bot$-child. Similarly, the second node corresponding to $\mathtt{y}$ has a $\bot$-child, since

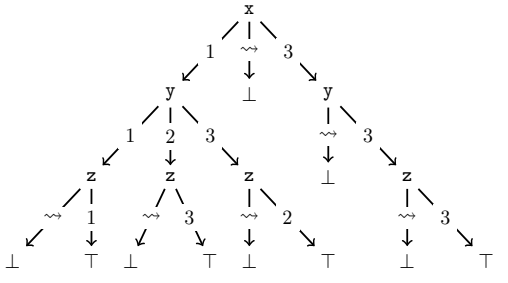


Figure 2 A prefix-tree reachability game arena.

there are no tuples in the interpretation of $r^\natural$ mapping $\mathtt{x}$ to 3 and $\mathtt{y}$ to 1 or 2. To complete the construction of the arena we only need to assign the nodes to the players. If a variable x is existentially (*resp.*, universally) quantified in $\wp$, the corresponding node of the tree is assigned to the first (*resp.*, second) player. Now, it is not hard to prove that $\mathcal{R}^\natural \models \wp \flat^\natural r^\natural$ iff the first player wins the reachability game to $\top$ on this trie. For instance, consider again the above arena and suppose that $\wp = \exists\mathtt{x}\forall\mathtt{y}\exists\mathtt{z}$. Then, $\mathcal{R}^\natural \models \wp \flat^\natural r^\natural$, as the first player has a winning strategy on the nodes $\mathtt{x}$ and $\mathtt{z}$. On the contrary, assume $\wp = \forall\mathtt{x}\exists\mathtt{y}\exists\mathtt{z}$. In this case $\mathcal{R}^\natural \not\models \wp \flat^\natural r^\natural$, since the second player can choose to reach the $\bot$-child of $\mathtt{x}$.

Summing up, since the problem of determining the winner of a turn-based reachability game is solvable in PTime [30], the complexity of our procedure immediately follows.

5 Satisfiability

We finally come to the more technical part of the paper, in which we provide the satisfiability procedure for FOL[1B]. In addition we describe the key model-theoretic tool at its basis. As mentioned before, this is focused on the concept of overlapping templates, *i.e.*, intuitively, a set $\{(\wp_i, \flat_i)\}$ of quantification and binding prefixes for which *(i)* there is a strict total order between the arguments that agrees, via $\flat_i$, with the functional dependences of all $\wp_i$ and *(ii)* each argument is bound to at most one existential variable. For the sake of space, here we just give the basic definitions that are necessary to formalize this concept and state the main characterization theorem. All proofs together with further notions and details will be reported in the extended version of this work.

In order to have a deeper intuition on the novel model-theoretic tool, we first describe four examples, built on the LS $\mathcal{L} = \langle\{\mathtt{a},\mathtt{b},\mathtt{c}\},\{\mathtt{q},\mathtt{r}\},\mathsf{ar}\rangle$ with $\mathsf{ar}(\mathtt{q}) = \mathsf{ar}(\mathtt{r}) = \{\mathtt{a},\mathtt{b},\mathtt{c}\}$, which cover some interesting cases of correlation between the satisfiability of a sentence and the overlapping property of its templates. Consider the sentence $\varphi_1 = \wp_1\flat_1\mathtt{q} \wedge \wp_2\flat_2\neg\mathtt{r} \wedge \wp_3\flat_3(\mathtt{q} \leftrightarrow \mathtt{r})$, where $\wp_1 = \forall\mathtt{x}\exists\mathtt{y}\forall\mathtt{z}$, $\wp_2 = \forall\mathtt{y}\exists\mathtt{z}\forall\mathtt{x}$, $\wp_3 = \forall\mathtt{x}\forall\mathtt{y}\forall\mathtt{z}$, and $\flat_1 = \flat_2 = \flat_3 = (\mathtt{a},\mathtt{x})(\mathtt{b},\mathtt{y})(\mathtt{c},\mathtt{z})$. It is not hard to see that φ_1 is unsatisfiable, since it requires the inconsistent derived relations $\mathtt{q}$, $\neg\mathtt{r}$, and $\mathtt{q} \leftrightarrow \mathtt{r}$ to hold on the same tuple function $\mathtt{t}$. Indeed, $\wp_1\flat_1\mathtt{q}$ forces $\mathtt{q}$ to hold on all tuples $\mathtt{t}_1$ satisfying the constraint $\mathtt{t}_1(\mathtt{b}) = \mathtt{f}_1(\mathtt{t}_1(\mathtt{a}))$, where $\mathtt{f}_1$ is a Skolem function for $\mathtt{y}$ in $\wp_1$. Similarly, $\wp_2\flat_2\neg\mathtt{r}$ demands $\neg\mathtt{r}$ on all tuples $\mathtt{t}_2$ with $\mathtt{t}_2(\mathtt{c}) = \mathtt{f}_2(\mathtt{t}_2(\mathtt{b}))$, where $\mathtt{f}_2$ is a Skolem function for $\mathtt{z}$ in $\wp_2$. Finally, $\wp_3\flat_3(\mathtt{q} \leftrightarrow \mathtt{r})$ enforces $\mathtt{q} \leftrightarrow \mathtt{r}$ on every possible tuple. Hence, the sentence φ_1 requires $\psi = \mathtt{q} \wedge \neg\mathtt{r} \wedge (\mathtt{q} \leftrightarrow \mathtt{r})$ on all tuples $\mathtt{t}$ with $\mathtt{t}(\mathtt{b}) = \mathtt{f}_1(\mathtt{t}(\mathtt{a}))$ and $\mathtt{t}(\mathtt{c}) = \mathtt{f}_2(\mathtt{t}(\mathtt{b}))$, which leads to an inconsistency. Observe that we can find such tuples because of the dependency order $\mathtt{a} \rightsquigarrow \mathtt{b} \rightsquigarrow \mathtt{c}$ among the arguments, which is compatible with the functional dependences of $\wp_1$ and $\wp_2$ via the bindings $\flat_1$ and $\flat_2$. Also, there are no arguments associated to more than one existential variable. Consequently the set of templates $\{(\wp_1,\flat_1),(\wp_2,\flat_2),(\wp_3,\flat_3)\}$ is said to be overlapping. Now, let φ_2 be the sentence obtained from φ_1 by replacing the prefixes $\wp_3$ and $\flat_3$ with $\forall\mathtt{x}\forall\mathtt{y}$ and $(\mathtt{a},\mathtt{x})(\mathtt{b},\mathtt{y})(\mathtt{c},\mathtt{y})$. We show that φ_2 is satisfiable on an RS of order $|\mathrm{Dm}| = 2$. First note that $\wp_3\flat_3(\mathtt{q} \leftrightarrow \mathtt{r})$ enforces $\mathtt{q} \leftrightarrow \mathtt{r}$ only on tuples $\mathtt{t}_3$ with $\mathtt{t}_3(\mathtt{b}) = \mathtt{t}_3(\mathtt{c})$. Moreover, choose a Skolem function $\mathtt{f}_2$ for $\mathtt{z}$ in $\wp_2$ such that $\mathtt{f}_2(\alpha) \neq \alpha$, for every $\alpha \in \mathrm{Dm}$, and suppose that φ_2 requires ψ on a tuple $\mathtt{t}$. In this case, we have that $\mathtt{t}(\mathtt{c}) = \mathtt{f}_2(\mathtt{t}(\mathtt{b}))$ and $\mathtt{t}(\mathtt{b}) = \mathtt{t}(\mathtt{c})$, *i.e.*, $\mathtt{t}(\mathtt{c}) = \mathtt{f}_2(\mathtt{t}(\mathtt{c}))$, which is impossible. Observe that there are no such tuples $\mathtt{t}$ because of a cyclic dependence $\mathtt{c} \rightsquigarrow \mathtt{c}$ caused by the second and third sentence. Therefore, the set of templates $\{(\wp_2,\flat_2),(\wp_3,\flat_3)\}$ is not overlapping. Similarly, consider the sentence φ_3 drawn from φ_1 by substituting $\forall\mathtt{z}\exists\mathtt{x}\forall\mathtt{y}$ for the prefix $\wp_3$. Also in this case, φ_3 is satisfiable on an RS of order 2. Indeed, suppose it requires ψ on a tuple $\mathtt{t}$. Then, we have that $\mathtt{t}(\mathtt{b}) = \mathtt{f}_1(\mathtt{t}(\mathtt{a}))$, $\mathtt{t}(\mathtt{c}) = \mathtt{f}_2(\mathtt{t}(\mathtt{b}))$, and $\mathtt{t}(\mathtt{a}) = \mathtt{f}_3(\mathtt{t}(\mathtt{c}))$, where $\mathtt{f}_3$ is a Skolem

function for $\mathtt{x}$ in $\wp_3$. Thus, $\mathsf{t}(\mathtt{a}) = \mathsf{f}_3(\mathsf{f}_2(\mathsf{f}_1(\mathsf{t}(\mathtt{a}))))$. Now, it is not hard to find f_1, f_2, and f_3 in such a way that $\mathsf{f}_3(\mathsf{f}_2(\mathsf{f}_1(\alpha))) \neq \alpha$, for every $\alpha \in \mathrm{Dm}$. So, the tuple t cannot exist, due to the cyclic dependence $\mathtt{a} \rightsquigarrow \mathtt{b} \rightsquigarrow \mathtt{c} \rightsquigarrow \mathtt{a}$. Here, $\{(\wp_1, \flat_1), (\wp_2, \flat_2), (\wp_3, \flat_3)\}$ is a set of non overlapping templates. Finally, derive the sentence φ_4 from φ_1, by setting the prefix $\wp_3$ to $\exists \mathtt{z} \forall \mathtt{x} \forall \mathtt{y}$. Again, φ_4 is satisfiable, as the argument $\mathtt{c}$ is existential in both $(\wp_2, \flat_2)$ and $(\wp_3, \flat_3)$. So, one can find a Skolem constant for $\mathtt{z}$ in $\wp_3$ that is different from all possible values assumed by the Skolem function f_2 for $\mathtt{z}$ in $\wp_2$.

Satisfiability Characterization. In this technical subsection, we state the fundamental characterization theorem connecting the satisfiability of a sentence with the overlapping property of its templates. To do this, we first give the formalization of the latter concept together with some auxiliary definition. Then, we introduce two suitable graphs defined on the structural features of a sets of templates, which are used to define the notion of overlapping.

A *template* $\tau \triangleq (\wp, \flat) \in \mathrm{Tem}$ is a pair of a quantification and a binding prefix on the set of arguments $\mathsf{ar}(\tau) \subseteq \mathrm{Ar}$ and variables $\mathsf{vr}(\tau) \subseteq \mathrm{Vr}$. By $\mathrm{Tem}(\mathrm{A})$ we denote the subset of templates having argument set $\mathrm{A} \subseteq \mathrm{Ar}$. For a given template $\tau = (\wp, \flat)$, by $\exists(\tau)$ (*resp.*, $\forall(\tau)$) we denote the set of arguments that are associated with an existential (*resp.*, universal) variable in $\wp$ via $\flat$. Moreover, $\cong_\tau \subseteq \mathsf{ar}(\tau) \times \mathsf{ar}(\tau)$ and $\rightsquigarrow_\tau \subseteq \forall(\tau) \times \exists(\tau)$ represent, respectively, the *collapsing equivalence* and the *functional dependence* induced by τ, *i.e.*, for all $a_1, a_2 \in \mathsf{ar}(\tau)$, $a_1 \cong_\tau a_2$ iff $\flat(a_1) = \flat(a_2)$ and, for all $a_1 \in \forall(\tau)$ and $a_2 \in \exists(\tau)$, $a_1 \rightsquigarrow_\tau a_2$ iff the variable $\flat(a_1)$ occurs before the variable $\flat(a_2)$ in $\wp$. For instance, for the template $\tau = (\forall \mathtt{x} \exists \mathtt{y}, (\mathtt{a}, \mathtt{x})(\mathtt{b}, \mathtt{y})(\mathtt{c}, \mathtt{x}))$, it holds that $\exists(\tau) = \{\mathtt{b}\}$, $\forall(\tau) = \{\mathtt{a}, \mathtt{c}\}$, $\mathtt{a} \rightsquigarrow_\tau \mathtt{b}$, $\mathtt{c} \rightsquigarrow_\tau \mathtt{b}$, and $\mathtt{a} \cong_\tau \mathtt{c}$.

As set of vertexes of the above mentioned graphs, we use the set $\mathrm{Ar}^{\mathrm{A}}_{\mathrm{S}} \subseteq \mathrm{S} \times \mathrm{Ar}$ of *extended arguments* $e = (\tau, a)$, *i.e.*, pairs of a template $\tau(e) = \tau$ and one of its arguments $a(e) = a \in \mathsf{ar}(\tau) \cap \mathrm{A}$. Also, $\exists^{\mathrm{A}}_{\mathrm{S}}$ (*resp.*, $\forall^{\mathrm{A}}_{\mathrm{S}}$) represents the set of existential (*resp.*, universal) extended arguments, *i.e.*, the elements e such that $a(e) \in \exists(\tau(e))$ (*resp.*, $a(e) \in \forall(\tau(e))$).

The *collapsing graph* for S over A is the symmetric directed graph $\mathcal{C}^{\mathrm{A}}_{\mathrm{S}} \triangleq \langle \mathrm{Ar}^{\mathrm{A}}_{\mathrm{S}}, \cong^{\mathrm{A}}_{\mathrm{S}} \rangle$ with the extended arguments as vertexes and the edge relation given by $\cong^{\mathrm{A}}_{\mathrm{S}} \triangleq \{(e_1, e_2) \in \mathrm{Ar}^{\mathrm{A}}_{\mathrm{S}} \times \mathrm{Ar}^{\mathrm{A}}_{\mathrm{S}} : (\tau(e_1) = \tau(e_2) \Rightarrow a(e_1) \cong_{\tau(e_1)} a(e_2)) \wedge (\tau(e_1) \neq \tau(e_2) \Rightarrow a(e_1) = a(e_2))\}^+$. Intuitively, $\cong^{\mathrm{A}}_{\mathrm{S}}$ is the least equivalence relation on $\mathrm{Ar}^{\mathrm{A}}_{\mathrm{S}}$ that identifies the arguments bound to the same variable in some template. This graph is used to take into account the multiple possible associations of an argument with the existential variables, in order to formalize the property *(ii)* described at the beginning of the section. In Figure 3, we depict the collapsing graphs $\mathcal{C}_i = \mathcal{C}^{\mathrm{A}}_{\mathrm{S}_i}$ for the set of templates $\mathrm{S}_i = \{\tau_1 = (\wp_1, \flat_1), \tau_2 = (\wp_2, \flat_2), \tau_3 = (\wp_3, \flat_3)\}$ and arguments $\mathrm{A} = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ associated with every single sentences φ_i given above, where $i \in \{1, 3, 4\}$. The dots simply represent the extended arguments obtained by intersecting rows and columns. In Figure 4, we report the collapsing graph $\mathcal{C}_2 = \mathcal{C}^{\mathrm{A}}_{\mathrm{S}_2}$ for the sentence φ_2, where the edge between the vertexes $(\tau_3, \mathtt{b})$ and $(\tau_3, \mathtt{c})$ is due to the binding $\flat_3$. Note that, since $\cong^{\mathrm{A}}_{\mathrm{S}}$ is an equivalence relation, we are omitting the transitive closure from the graphs.

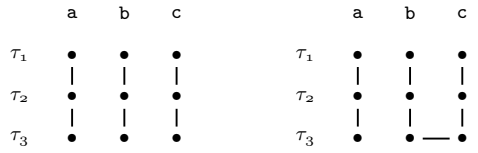


Figure 3 Collapsing graphs $\mathcal{C}_1$, $\mathcal{C}_3$, and $\mathcal{C}_4$.

Figure 4 Collapsing graph $\mathcal{C}_2$.

We can now define a property describing the case in which two existential arguments are forced to assume the same value. A collapsing graph $\mathcal{C}^{\mathrm{A}}_{\mathrm{S}}$ is *conflicting* iff there are two existential extended arguments $e_1, e_2 \in \exists^{\mathrm{A}}_{\mathrm{S}}$ such that $e_1 \cong^{\mathrm{A}}_{\mathrm{S}} e_2$ and, if $\tau(e_1) = \tau(e_2)$ then

$a(e_1) \not\cong_{\tau(e_1)} a(e_2)$. Intuitively, this property ensures the existence of two arguments that, at the same time, need to assume the same value, due to the collapsing equivalence on some template, and are both existentially quantified in possibly different templates. As an example, the collapsing graph $\mathcal{C}_2$ of φ_2 is conflicting, since $(\tau_1, \mathsf{b}) \cong^{\mathrm{A}}_{\mathrm{S}} (\tau_2, \mathsf{c})$. The same holds for the collapsing graph $\mathcal{C}_4$ of φ_4, since $(\tau_2, \mathsf{c}) \cong^{\mathrm{A}}_{\mathrm{S}} (\tau_3, \mathsf{c})$.

The second graph we introduce keeps track of the functional dependences between arguments that cross all the templates. This property has been informally described in point *(i)* at the beginning of this section. The *dependence graph* for S over A is the directed graph $\mathcal{D}^{\mathrm{A}}_{\mathrm{S}} \triangleq \langle \mathrm{Ar}^{\mathrm{A}}_{\mathrm{S}}, \rightsquigarrow^{\mathrm{A}}_{\mathrm{S}} \rangle$ with the extended arguments as vertexes and the edge relation given by $\rightsquigarrow^{\mathrm{A}}_{\mathrm{S}} \triangleq \cong^{\mathrm{A}}_{\mathrm{S}} \circ \{(e_1, e_2) \in \mathrm{Ar}^{\mathrm{A}}_{\mathrm{S}} \times \mathrm{Ar}^{\mathrm{A}}_{\mathrm{S}} : \tau(e_1) = \tau(e_2) \wedge a(e_1) \rightsquigarrow_{\tau(e_1)} a(e_2)\}$. In Figures 5, 6, and 7, we report the dependence graphs $\mathcal{D}_i = \mathcal{D}^{\mathrm{A}}_{\mathrm{S}_i}$ corresponding to the sentences φ_1, φ_2, and φ_3, respectively. The black arrows represent the functional dependences inside a single template, while the gray ones are obtained by the composition with the collapsing relation. Note that $\mathcal{D}_1$ is acyclic, so, we can build an order among the extended arguments that agrees with all functional dependences of the templates. In $\mathcal{D}_2$, instead, there is a loop on (τ_2, c) due to the structures of τ_2 and τ_3. Finally, $\mathcal{D}_3$ contains a cycle among (τ_3, a), (τ_1, b), and (τ_2, c).

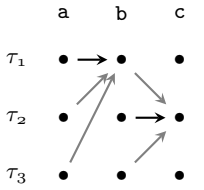


Figure 5 Dependence graph $\mathcal{D}_1$.

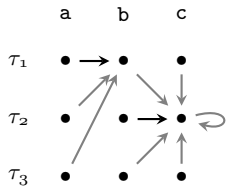


Figure 6 Dependence graph $\mathcal{D}_2$.

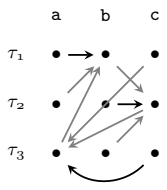


Figure 7 Dependence graph $\mathcal{D}_3$.

▶ **Definition 5.1** (Overlapping Templates). A set of templates $\mathrm{S} \subseteq \mathrm{Tem}(\mathrm{A})$ over a set of arguments $\mathrm{A} \subseteq \mathrm{Ar}$ is *overlapping* iff the collapsing graph $\mathcal{C}^{\mathrm{A}}_{\mathrm{S}}$ is not conflicting and the dependence graph $\mathcal{D}^{\mathrm{A}}_{\mathrm{S}}$ is acyclic.

Observe that there are similarities between the introduced concept of overlapping templates and notion of *weakly acyclic dependencies* of TGDs [41], which is known to be a sufficient property for the termination of the chase algorithm [1], one of the most useful tools in database theory to test query containment under constraints. This connection will be analysed in the journal version of this article.

We can finally state the characterization theorem. To do this, we first recall that, for a given set X and a Boolean formula η over X, we denote by $\mathsf{wit}(\eta) \subseteq 2^{\mathrm{X}}$ the set of *witnesses* of η, *i.e.*, the set of all possible subsets of X satisfying η.

▶ **Theorem 5.2** (Satisfiability Characterization). *A given* FOL[1B] *sentence φ is satisfiable iff there exists a witness $\mathrm{F} \in \mathsf{wit}(\varphi)$ such that, for all overlapping templates $\mathrm{S} \subseteq \mathsf{dom}(\mathsf{fr}) \cap \mathrm{Tem}(\mathrm{A})$ with $\mathrm{A} \subseteq \mathrm{Ar}$, it holds that $\bigwedge_{\tau \in \mathrm{S}} \mathsf{fr}(\tau)$ is a satisfiable Boolean combination of relations, where $\mathsf{fr} = \{(\wp, \flat) \in \mathrm{Tem} \mapsto \widehat{r} \in \widehat{\mathrm{Rl}} : \wp \flat \widehat{r} \in \mathrm{F}\}$ is the function associating each template with the corresponding derived relation in* F [1].

Satisfiability Procedure. We finally provide an algorithm for the solution of the satisfiability problem for FOL[1B], which can be interpreted as a satisfiability-modulo-theory procedure.

[1] Recall that a FOL[1B] sentence can be seen as a Boolean combination of sentences of the form $\wp \flat \widehat{r}$. Moreover, we are assuming that each quantification/binding prefix $\wp \flat$ in φ occurs only once, thus, fr is indeed a function (this can be ensured by a standard renaming of the variables).

Indeed, by means of a syntactic preprocessing based on the concept of overlapping templates, the search for a model of a FOL[1B] sentence is reduced to that of a sequence of Boolean formulas over the set of derived relations. The correctness of such an approach is crucially based on the fundamental characterization of Theorem 5.2. It is also interesting to observe that the procedure is independent from the size of the finite model derived in the proof of Theorem 3.11.

To understand the main idea behind the algorithm, it is useful to describe it through a simple three-round two-player turn-based game between the existential player, called Eloise, willing to show that a sentence φ is satisfiable, and the universal player, called Abelard, trying to do exactly the opposite. First, Eloise chooses a witness $\mathrm{F} \in \mathsf{wit}(\varphi)$ for φ seen as a Boolean combination of simpler subsentences of the form $\wp\flat\widehat{r}$. In this way, she identifies a formula function $\mathsf{fr} = \{(\wp,\flat) \in \mathrm{Tem} \mapsto \widehat{r} \in \widehat{\mathrm{Rl}} : \wp\flat\widehat{r} \in \mathrm{F}\}$ that describes F by associating each derived relation with the corresponding template. Then, Abelard chooses a subset of overlapping templates $\mathrm{S} \subseteq \mathsf{dom}(\mathsf{fr}) \cap \mathrm{Tem(A)}$ over a set of arguments $\mathrm{A} \subseteq \mathrm{Ar}$. At this point, Eloise wins the play iff the Boolean formula $\psi = \bigwedge_{\tau \in \mathrm{S}} \mathsf{fr}(\tau)$, obtained as the conjunction of all the derived relations associated with the templates in S, is satisfiable. If this is not the case, Abelard has spotted a subset $\{\wp\flat\mathsf{fr}(\tau) : \tau = (\wp,\flat) \in \mathrm{S}\}$ of the witness F that requires to verify the unsatisfiable property ψ on a certain valuation of arguments in A. Thus, F cannot have a model. Consequently, φ is satisfiable iff Eloise has a winning strategy for this game.

Algorithm 1: FOL[1B] Satisfiability Checker.

```
   signature sat : L(Vr)-FOL[1B] → {⊤, ⊥}
   function sat(φ)
1      foreach F ∈ wit(φ) do
2          fr ← {(℘, ♭) ∈ Tem ↦ r̂ ∈ R̂l : ℘♭r̂ ∈ F}
3          i ← ⊥
4          foreach S ⊆ dom(fr) ∩ Tem(A) with A ⊆ Ar do
5              if S is-overlapping-over A then
6                  if wit(⋀_{τ∈S} fr(τ)) = ∅ then
7                      i ← ⊤
8          if i = ⊥ then
9              return ⊤
10     return ⊥
```

We can now describe the pseudo-code of Algorithm 1. The deterministic counterpart of Eloise's choice is the selection of a witness $\mathrm{F} \in \mathsf{wit}(\varphi)$ in the loop at Line 1, which is followed by the computation of the corresponding formula function fr. At each iteration, a flag i is also set to $\bot$, with the aim to indicate that F is not inconsistent (a witness is consistent until proven otherwise). After this, we find the deterministic counterpart of Abelard's choice, implemented by the combination of a loop and a conditional statement at Lines 4 and 5, where a subset of overlapping templates $\mathrm{S} \subseteq \mathsf{dom}(\mathsf{fr}) \cap \mathrm{Tem(A)}$ over a set of arguments $\mathrm{A} \subseteq \mathrm{Ar}$ is selected. This is done in order to verify the inconsistency of the conjunction $\bigwedge_{\tau \in \mathrm{S}} \mathsf{fr}(\tau)$ at Line 6. If this check is positive then the flag i is switched to $\top$. Once all choices for Abelard are analysed, the computation reaches Line 8, where it is verified whether an inconsistency was previously found. In the negative case, the algorithm terminates by returning $\top$, with the aim to indicate that a good guess for Eloise is possible. In the case all witnesses are analysed, finding for each of them an inconsistency, Eloise has no winning strategy. Thus, the algorithm ends by returning $\bot$.

It remains to evaluate the complexity of the algorithm *w.r.t.* the length of the sentence φ. The verification at Lines 5-7 of Abelard's universal guess of Line 4 can be done in PTime with an NPTime advice for the Boolean satisfiability problem of Line 6. Thus, the check at Lines 2-

9 of Eloise's existential guess of Line 1 can be done in PTime with a CoNPTime$^{\text{NPTime}}$ advice for Line 4. Hence, the overall complexity is $\Sigma_3^{\text{P}} = \text{NPTime}^{\text{CoNPTime}^{\text{NPTime}}}$, *i.e.*, the problem belongs to the third level of the polynomial hierarchy.

6 Discussion

Trying to understand the reasons why some powerful extensions of modal logic are decidable, we introduced and studied a new family of FOL fragments based on the combinations of binding forms admitted in their formulas. In other words, we provided a novel criterion to classify FOL sentences focused on the associations between arguments and variables. A main features of this classification is to avoid the usual syntactic restrictions on quantifier patterns, number of variables, relation arities, and relativisation of quantifiers or negations. Therefore, it represents a suitable framework in which to study model-theoretic and algorithmic properties of extensions of modal logic, such as ATL* and SL[1G]. We analysed the expressiveness of the introduced fragments, showing that the simplest one, called *one binding* (FOL[1B]), is already incomparable with other important restrictions of FOL, such as the *clique guarded* (FOL[CG]) [19, 22], the *guarded negation* (FOL[GN]) [7], and the *fluted logic* (FOL[FL]) [44]. Moreover, we proved that it enjoys the finite-model property, a PTime model checking, a Σ_3^{P}-complete satisfiability, and a constructive version of both Craig's interpolation and Beth's definability, which are indirectly derived from the same properties of propositional logic. As future work, besides a deeper study of the *conjunctive* and *disjunctive* fragments (FOL[CB] and FOL[DB]), it is important to verify which of the stated results lift to the extensions of the introduced logics that comprise distinguished relations representing equivalences, orders, or the equality. Finally, it would be interesting to come up with a wider framework, which can accommodate the incomparable languages FOL[CG], FOL[GN], FOL[FL], and FOL[1B].

Acknowledgements We are very grateful to M. Benerecetti, L. Sauro, and M.Y. Vardi for many useful comments and discussions on a first draft of this work.

References

1 S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1995.
2 R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-Time Temporal Logic. *JACM*, 49(5):672–713, 2002.
3 H. Andréka, J. van Benthem, and I. Németi. Modal Languages And Bounded Fragments Of Predicate Logic. *JPL*, 27(3):217–274, 1998.
4 V. Bárány, M. Benedikt, and B. ten Cate. Rewriting Guarded Negation Queries. In *MFCS'13*, LNCS 8087, pages 98–110. Springer, 2013.
5 V. Bárány, G. Gottlob, and M. Otto. Querying the Guarded Fragment. In *LICS'10*, pages 1–10. IEEE Computer Society, 2010.
6 V. Bárány, B. ten Cate, and M. Otto. Queries with Guarded Negation. *PVLDB*, 5(11):1328–1339, 2012.
7 V. Bárány, B. ten Cate, and L. Segoufin. Guarded Negation. In *ICALP'11*, LNCS 6756, pages 356–367. Springer, 2011.
8 M. Ben-Ari. *Mathematical Logic for Computer Science (3rd ed.).* Springer, 2012.
9 D. Berwanger and E. Grädel. Games and Model Checking for Guarded Logics. In *LPAR'01*, LNCS 2250, pages 70–84. Springer, 2001.
10 E. Börger. Decision Problems in Predicate Logic. In *LC'82*, volume 112, pages 263–301. Elsevier, 1984.

11 E. Börger, E. Grädel, and Y. Gurevich. *The Classical Decision Problem.* Perspectives in Mathematical Logic. Springer, 1997.

12 K. Chatterjee, T.A. Henzinger, and N. Piterman. Strategy Logic. In *CONCUR'07*, LNCS 4703, pages 59–73. Springer, 2007.

13 E.F. Codd. A Relational Model of Data for Large Shared Data Banks. *CACM*, 13(6):377–387, 1970.

14 E.F. Codd. Relational Completeness of Data Base Sublanguages. *DS*, pages 65–98, 1972.

15 H.-D. Ebbinghaus and J. Flum. *Finite Model Theory.* Perspectives in Mathematical Logic. Springer, 1995.

16 R. Fagin and M.Y. Vardi. The Theory of Data Dependencies – A Survey. *MIP*, 34:19–71, 1972.

17 E. Fredkin. Trie Memory. *CACM*, 3(9):490–499, 1960.

18 E. Goncalves and E. Grädel. Decidability Issues for Action Guarded Logics. In *DL'00*, pages 123–132, 2000.

19 E. Grädel. Decision Procedures for Guarded Logics. In *CADE'99*, LNCS 1632, pages 31–51. Springer, 1999.

20 E. Grädel. On The Restraining Power of Guards. *JSL*, 64(4):1719–1742, 1999.

21 E. Grädel. Why are Modal Logics so Robustly Decidable? In *CTTCS'01*, pages 393–408. World Scientific, 2001.

22 E. Grädel. Guarded Fixed Point Logics and the Monadic Theory of Countable Trees. *TCS*, 288(1):129–152, 2002.

23 E. Grädel. Decidable Fragments of First-Order and Fixed-Point Logic. In *KWLCS'03*, 2003.

24 E. Grädel, P.G. Kolaitis, and M.Y. Vardi. On the Decision Problem for Two-Variable First-Order Logic. *BSL*, 3(1):53–69, 1997.

25 E. Grädel and I. Walukiewicz. Guarded Fixed Point Logic. In *LICS'99*, pages 45–54. IEEE Computer Society, 1999.

26 Lauri Hella and A. Kuusisto. One-Dimensional Fragment of First-Order Logic. In *AIML'14*, pages 274–293. College Publications, 2014.

27 D. Hilbert and W. Ackermann. *Grundzüge der Theoretischen Logik.* Die Grundlehren der Mathematischen Wissenschaften. Springer, 1928.

28 I. Hodkinson. Loosely Guarded Fragment of First-Order Logic has the Finite Model Property. *SL*, 70(2):205–240, 2002.

29 E. Hoogland and M. Marx. Interpolation and Definability in Guarded Fragments. *SL*, 70(3):373–409, 2002.

30 N. Immerman. Number of Quantifiers is Better Than Number of Tape Cells. *JCSS*, 22(3):384–406, 1981.

31 N. Immerman. Relational Queries Computable in Polynomial Time (Extended Abstract). In *STOC'82*, pages 147–152. Association for Computing Machinery, 1982.

32 N. Immerman. Relational Queries Computable in Polynomial Time. *IC*, 68(1-3):86–104, 1986.

33 N. Immerman. *Descriptive Complexity.* Graduate Texts in Computer Science. Springer, 1999.

34 E. Kieronski and A. Kuusisto. Complexity and Expressivity of Uniform One-Dimensional Fragment with Equality. In *MFCS'14*, LNCS 8634, pages 365–376. Springer, 2014.

35 L. Löwenheim. Über Möglichkeiten im Relativkalkül. *MA*, 76(4):447–470, 1915.

36 F. Mogavero, A. Murano, G. Perelli, and M.Y. Vardi. Reasoning About Strategies: On the Satisfiability Problem. Under submission.

37 F. Mogavero, A. Murano, G. Perelli, and M.Y. Vardi. What Makes ATL* Decidable? A Decidable Fragment of Strategy Logic. In *CONCUR'12*, LNCS 7454, pages 193–208. Springer, 2012.

38 F. Mogavero, A. Murano, G. Perelli, and M.Y. Vardi. Reasoning About Strategies: On the Model-Checking Problem. *TOCL*, 15(4):34:1–42, 2014.

39 F. Mogavero, A. Murano, and M.Y. Vardi. Reasoning About Strategies. In *FSTTCS'10*, LIPIcs 8, pages 133–144. Leibniz-Zentrum fuer Informatik, 2010.

40 M. Mortimer. On Languages with Two Variables. *MLQ*, 21(1):135–140, 1975.

41 A. Onet. The Chase Procedure and its Applications in Data Exchange. In *Data Exchange, Integration, and Streams.*, pages 1–37. Leibniz-Zentrum fuer Informatik, 2013.

42 W.C. Purdy. Complexity and Nicety of Fluted Logic. *SL*, 71(2):177–198, 2002.

43 W.C. Purdy. Inexpressiveness of First-Order Fragments. *AJL*, 4:1–12, 2006.

44 W.V. Quine. Variables Explained Away. *PAPS*, 104(3), 1960.

45 I. Sain. Beth's and Craig's Properties via Epimorphisms and Amalgamation in Algebraic Logic. In *ALUACS'88*, LNCS 425, pages 209–225. Springer, 1990.

46 S. Schewe. ATL* Satisfiability is 2ExpTime-Complete. In *ICALP'08*, LNCS 5126, pages 373–385. Springer, 2008.

47 S. Schewe and B. Finkbeiner. Satisfiability and Finite Model Property for the Alternating-Time muCalculus. In *CSL'06*, pages 591–605. Springer, 2006.

48 S.G. Simpson. Partial Realizations of Hilbert's Program. *JSL*, 53(2):349–363, 1988.

49 B. ten Cate and L. Segoufin. Unary Negation. In *STACS'11*, LIPIcs 9, pages 344–355. Leibniz-Zentrum fuer Informatik, 2011.

50 B. ten Cate and L. Segoufin. Unary Negation. *LMCS*, 9(3):1–46, 2013.

51 J.D. Ullman. *Principles of Database Systems.* Computer Science Press, 1982.

52 J. van Benthem. Dynamic Bits And Pieces. Technical report, University of Amsterdam, Amsterdam, Netherlands, 1997.

53 J. van Benthem. Modal Logic In Two Gestalts. In *AIML'98*, pages 91–118. CSLI Publications, 2000.

54 D. van Dalen. *Logic and Structure (3rd ed.).* Universitext. Springer, 1994.

55 M.Y. Vardi. The Complexity of Relational Query Languages (Extended Abstract). In *STOC'82*, pages 137–146. Association for Computing Machinery, 1982.

56 M.Y. Vardi. On the Complexity of Bounded-Variable Queries. In *PODS'95*, pages 266–276. Association for Computing Machinery, 1995.

57 M.Y. Vardi. Why is Modal Logic So Robustly Decidable? In *DCFM'96*, pages 149–184. American Mathematical Society, 1996.

58 D. Walther, C. Lutz, F. Wolter, and M. Wooldridge. ATL Satisfiability is Indeed ExpTime-Complete. *JLC*, 16(6):765–787, 2006.